robotron

Programmtechnische
Beschreibung

Programmiersystem Standard-BASIC

C 3016-0300-1 M 3030

Arbeitsplatzcomputer A 7150 Betriebssystem DCP

ANWENDER-DOKUMENTATION	Standard-BASIC	MOS
10/87	Sprachbeschreibung	

Programmtechnische Beschreibung

MOS
Programmiersystem
Standard-BASIC

VEB Robotron-Projekt Dresden

C3016-0300-1 M3030

Die vorliegende Anwenderdokumentation "Sprachbeschreibung Standard-BASIC" entspricht dem Stand von 10/87.

Nachdruck, jegliche Vervielfaeltigung oder Auszuege daraus sind unzulaessig.

Herausgeber:
VEB Robotron-Projekt Dresden
Leningrader Strasse 9
Dresden
8 0 1 0

C 1987 robotron

Kurzreferat
In der vorliegenden Sprachbeschreibung werden die Ausdrucksmittel der Programmiersprache BASIC definiert. Die Sprache basiert auf dem ANSI-Standard X3.113 1987. BASIC ist fuer unterschiedliche Anwendungsfaelle geeignet. Die Handhabung des Systems wird in der Anleitung fuer den Bediener beschrieben.

Inhaltsverzeichnis

		Seite
1.	Einleitung	5
2.	Methoden der Sprachdefinition	6
3.	Grundlegende Definitionen	9
3.1.	Zeichen	9
3.2.	Schluesselworte	10
3.3.	Reservierte Worte	11
3.4.	Identifikatoren	12
4.	Grundlegende Programmstrukturen	14
4.1.	Programmeinheiten	14
4.2.	Zeilen, Kommentarzeilen	14
4.3.	Struktur des Hauptprogramms	16
5.	Verarbeitung numerischer Werte	19
5.1.	Numerische Konstanten	19
5.2.	Numerische Variable	20
5.3.	Numerische Ausdruecke	21
5.4.	Numerische Standardfunktionen	24
5.5.	Let-Anweisung fuer numerische Werte	27
5.6.	Numerische Genauigkeit, Winkeldarstellung und Deklaration von numerischen Variablen	28
6.	Verarbeitung von String-Werten	30
6.1.	String-Konstanten	30
6.2.	String-Variable	30
6.3.	String-Ausdruecke	32
6.4.	String-Standardfunktionen	34
6.5.	Let-Anweisung fuer String-Werte	37
6.6.	Deklaration von String-Variablen	38
7.	Felder	40
7.1.	Deklaration von Feldern	40
8.	Steueranweisungen und Steuerstrukturen	44
8.1.	Logische Ausdruecke	44
8.2.	Steueranweisungen	47
8.3.	Schleifen	50
8.4.	If-Anweisung und Fallauswahl	54
8.4.1.	If-Anweisung und If-Block	54
8.4.2.	Fallauswahl	56
9.	Unterprogrammtechnik und Programmverkettung	59
9.1.	Funktionen	59
9.1.1.	Funktionsdefinition	59
9.1.2.	Spezifikation von Funktionen	61
9.1.3.	Parameteruebergabe, Berechnung des Funktionswertes	62
9.2.	Subroutinen	64
9.2.1.	Subroutinedefinition	64
9.2.2.	Spezifikation von Subroutinen	65
9.2.3.	Parameteruebergabe, CALL-Anweisung	66
9.3.	Programmverkettung	69

10. Ein- und Ausgaben 71
10.1. Eingabe aus der internen Datenliste 71
10.1.1. Data-Anweisung 71
10.1.2. Read-Anweisung 72
10.1.3. Restore-Anweisung 73
10.2. Eingabe vom Terminal 74
10.3. Nichtformatierte Ausgabe ueber das Terminal 77
10.3.1. Aufbau der Ausgabezeile 78
10.3.2. Veraenderung und Abfrage der Zeilenbreite und der Zonenlaenge 79
10.3.3. Abarbeitung der Druckelemente 80
10.3.4. Abarbeitung der Drucktrenner 82
10.4. Formatierte Ausgabe ueber das Terminal 83
10.4.1. Abarbeitung der print-using-Anweisung 84
10.4.2. Aufbereitung numerischer Werte 86
10.4.3. Aufbereitung von Stringwerten 90
10.4.4. Ausnahmen bei formatierter Terminalausgabe 91

11. Dateien 92
11.1. Dateioperationen 93
11.1.1. Open-Anweisung 93
11.1.2. Close-Anweisung 100
11.1.3. Erase-Anweisung 100
11.1.4. Ask-Anweisung 101
11.2. Positionierung des Dateizeigers 105
11.3. Erzeugung von Dateien 109
11.3.1. Genereller Ablauf einer Schreiboperation 109
11.3.2. Print-Anweisung 109
11.3.3. Set-Anweisung 112
11.3.4. Write-Anweisung 113
11.4. Eingabe aus Dateien 115
11.4.1. Genereller Ablauf einer Eingabeoperation 115
11.4.2. Input-Anweisung 117
11.4.3. Read-Anweisung 120

12. Ausnahmebehandlung und Testunterstuetzung 125
12.1. Ausnahmebehandlung 125
12.2. Testunterstuetzung 129

Anlage 1: Erweiterungen 132
Anlage 2: Zeichensatz 138
Anlage 3: Ausnahmecodes 141

Tabellenverzeichnis

Tabelle 1: Standardfunktionen 24
Tabelle 2: String-Standardfunktionen 35
Tabelle 3: Verwendung der Formatzeichen 88
Tabelle 4: Dateiorganisation und Datensatzart 92
Tabelle 5: Dateieigenschaften 103
Tabelle 6: Positionierung des Dateizeigers nach Ausnahmen bei Eingabeoperationen 116
Tabelle 7: Typ des Resultates einer arithmetischen Operationen 134

1. Einleitung

Die Grundlage der in der vorliegenden Dokumentation beschriebenen Sprache BASIC ist der BASIC-Standard ANSI X3.113 1987. Dieser Standard definiert einen Sprachkern und 6 Erweiterungsmoduln. Der Sprachkern enthaelt Mittel zur Definition und Verarbeitung von numerischen und String-Daten, zur Bildung von Steuerstrukturen, zur Realisierung von sequentiellen Ein- und Ausgaben, zur Ausnahmebehandlung und zur Unterprogrammtechnik. Mit dem Sprachkern werden im wesenlichen alle Ausdrucksmoeglichkeiten bereitgestellt, die in den verschiedenen, verbreiteten BASIC-Versionen ueblich sind. Die 6 Erweiterungsmoduln umfassen eine Editorerweiterung, eine Festkommaarithmetik, zwei Ein- und Ausgabeerweiterungen, Anweisungen zur Grafik und Anweisungen zur Echtzeitarbeit.
Die in der vorliegenden Dokumentation beschriebene Sprache BASIC umfasst den Sprachkern mit Ausnahme der sog. MAT-Anweisungen und die Editorerweiterung. Die Editorerweiterung ist in der "Anleitung fuer den Bediener" beschrieben. Vom Standard wurde nur dort abgewichen, wo die Ressourcen fuer eine standardgerechte Implementierung nicht ausreichten. Die wenigen Abweichungen vom Standard sind geringfuegig. Sie sind im Text explizit ausgewiesen. Gegenueber dem Standard wurde eine Erweiterung vorgenommen, die eine Arbeit mit Integer-Groessen zulaesst. Diese Erweiterung ist syntaktisch so formuliert, dass eine Ueberfuehrung in ein standardgerechtes Programm ohne Probleme moeglich ist.

2. Methoden der Sprachdefinition

Die folgenden Abschnitte dieses Handbuchs haben einen einheitlichen Aufbau:

- Der Beginn jedes Abschnittes enthaelt einen Ueberblick ueber die im Abschnitt behandelten Sprachelemente.
- Unter dem Stichwort "Syntax" wird der korrekte Aufbau der behandelten Sprachelemente aus anderen Sprachelementen oder Zeichen des Alphabets beschrieben. Diese Beschreibung erfolgt durch Syntaxregeln.
- Nach der Syntaxbeschreibung folgen Angaben zur Bedeutung der Sprachelemente (Semantik) und zu ihrer zweckmaessigen Verwendung (Pragmatik). Beispiele ergaenzen diesen Teil und erleichtern das Erlernen der Sprache.

In der Syntaxbeschreibung wird jede Sprachkonstruktion durch eine metalinguistische Variable bezeichnet. Die Bezeichnungen der metalinguistischen Variablen sind so gewaehlt, dass sie auf die Bedeutung der Variablen hinweisen. Metalinguistische Variablen sind klein geschrieben. Bestehen sie aus mehreren Worten, sind diese durch das Unterstreichungszeichen verbunden. In den Syntaxregeln werden metalinguistische Variable und Zeichen aus dem Alphabet von BASIC durch metalinguistische Konnektoren verknuepft:

metalinguistischer Konnektor	Verwendung als
::=	Definitionszeichen
|	Alternativzeichen
$[\ ]^{n}_{t}$	Alternativ- u. Wiederholungskonstruktion fuer die 0 bis n-malige Wiederholung
$\{\ \}^{n}_{t}$	Alternativ- u. Wiederholungskonstruktion fuer die 1 bis n-malige Wiederholung

In den beiden Wiederholungskonstruktionen bezeichnet n die maximale Anzahl von Wiederholungen. Ist n nicht angegeben, gilt n = 1. Ist fuer n das Zeichen * angegeben, ist die Zahl der Wiederholungen nicht begrenzt. t bezeichnet das Trennzeichen, mit dem die wiederholten Konstruktionen zu trennen sind. Fehlt dieses Trennzeichen, werden die wiederholten Konstruktionen nicht voneinander getrennt.
In einer Syntaxregel steht links vom Definitionszeichen immer eine metalinguistische Variable, die durch die rechts vom Definitionszeichen stehende Folge von Zeichen, metalinguistischen Variablen und Konnektoren definiert wird. Die Verwendung der metalinguistischen Konnektoren wird an den folgenden Beispielen erklaert:

- Folgen von metalinguistischen Variablen und Zeichen:

```
formaler_kanal ::= *ganze_zahl
```

Diese Syntaxregel aus Abschnitt 9.2.3. definiert, dass ein formaler Kanal durch das Zeichen "*" gefolgt von einer ganzen Zahl dargestellt wird. Der Aufbau einer ganzen Zahl ist durch die Syntaxregel definiert, in der "ganze_zahl" links vom Definitonszeichen auftritt. Diese Syntaxregel befindet sich im Abschnitt 5.1.

- Wiederholungskonstruktion mit geschweiften Klammern:

$$\text{ganze\_zahl} ::= \{\text{ziffer}\}^{*}$$

Diese Syntaxregel besagt, dass eine ganze Zahl durch eine Wiederholung von Ziffern dargestellt wird. Es ist mindestens eine Ziffer anzugeben (geschweifte Klammer) und die Zahl der Ziffern ist nicht begrenzt (n = *). Zwischen den Ziffern steht kein Trennzeichen (t nicht angeben).

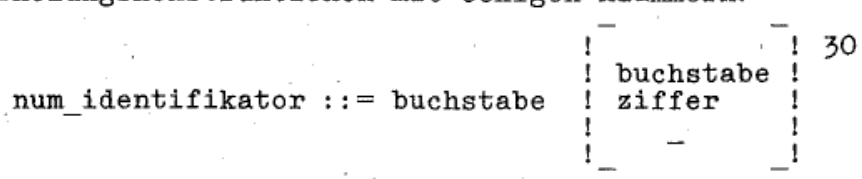
- Alternativkonstruktion:

Die metalinguistische Variable "ziffer" ist in Abschnitt 3.1. definiert.

$$\text{ziffer} ::= 0|1|2|3|4|5|6|7|8|9$$

Das Alternativzeichen besagt hier, dass "ziffer" entweder durch das Alphabetzeichen 0 oder 1 oder 2 oder ... dargestellt werden kann. In eckigen oder geschweiften Klammern wird das Alternativzeichen nicht verwendet. Dort werden die alternativ zu verwendenden Symbole oder Symbolfolgen untereinander notiert (siehe folgendes Beispiel).

- Wiederholungskonstruktionen mit eckigen Klammern:



Ein numerischer Indentifikator beginnt mit einem Buchstaben. Es folgen 0 bis 30 Buchstaben oder Ziffern oder Unterstreichungszeichen. Korrekte numerische Identifikatoren waeren also z.B.: F, FB, F7, Feld_grenze_1, a4_format.

Bei der Sprachbeschreibung gibt es folgende Besonderheiten:

- Kann die Symbolfolge auf der rechten Seite einer Syntaxregel nicht auf einer Zeile untergebracht werden, wird die rechte Seite der Regel eingerueckt unter die linke Seite gesetzt und wenn noetig auf mehrere Zeilen geteilt.
- Um den Suchaufwand bei der Arbeit mit der Syntax zu verringern, sind einzelne Regeln gleichlautend in verschiedenen Abschnitten enthalten (z.B. die Regel fuer "relation" in Abschnitt 8.1 und Abschnitt 8.4.2.)
- Bestimmte metalinguistische Variable sind durch mehrere von einander verschiedene Syntaxregeln definiert (so z.B. typ_deklaration durch Syntaxregeln in Abschnitt 5.6., 6.6., 9.1.2. und 9.2.2.). Jede dieser Regeln definiert diese metalinguistische Variable alternativ zu den uebrigen Regeln.

Anwendung der Syntaxregeln:

Wendet man die ersten drei der in den Beispielen enthaltenen Syntaxregeln der Reihe nach an, in dem man jeweils metalinguistische Variablen durch eine ihrer Definition entsprechende Symbolfolge ersetzt, so kann man z.B. aus "formaler Kanal" ueber:

```
formaler_kanal -> *ganze_zahl -> *ziffer ziffer -> *07
```

die Zeichenreihe "*07" entwickeln. Diese Zeichenfolge waere eine korrekte Notation eines formalen Kanals. Diese Art der Anwendung von Syntaxregeln klaert Fragen, welche Zeichen und welche Sprachelemente in einer bestimmten Konstruktion formal zulaessig sind bzw. miteinander "kombiniert" werden duerfen.

3. Grundlegende Definitionen

Jedes BASIC-Programm besteht aus einer Folge von Zeilen, die Anweisungen enthalten. Jede Zeile ist eine Folge von Zeichen.

3.1. Zeichen

Der Zeichensatz von BASIC ist eine Teilmenge des ASCII-Zeichensatzes gemaess ANSI-Standard ANSI X3.4.-1977.

Syntax

```
               |-                 -|
               |  buchstabe        |
zeichen ::= <  |  ziffer           |>
               |  sonderzeichen    |
               |_                 _|

                 |-                  -|
                 |  grossbuchstabe    |
buchstabe ::= <  |                    |>
                 |  kleinbuchstabe    |
                 |_                  _|

grossbuchstabe ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|
                   X|Y|Z

kleinbuchstabe ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|
                   x|y|z

ziffer ::= 0|1|2|3|4|5|6|7|8|9

sonderzeichen ::= "|&|'|*|^|:|,|¤|=|!|>|(|<|•|%|
                  ?|)|;|/|_|.|+|-|leerzeichen|
```

Der ASCII-Zeichensatz ist in Anlage 2 enthalten. Fuer alle Zeichen werden die Codes und soweit vorhanden die graphische Darstellung , der mnemonische Name und der Name angegeben. Durch die numerischen Werte der Codes wird eine Reihenfolge der Zeichen definiert, auf der der Vergleich von Zeichen bzw. Zeichenketten beruht.
Korrespondierende Gross- und Kleinbuchstaben sind aequivalent, wenn sie in Identifikatoren und Schluesselwoertern auftreten.
So kann z.B. das Schluesselwort PRINT auch als

Print oder print

gschrieben werden.
Leerzeichen koennen mit wenigen Ausnahmen ueberall in einem BASIC-Programm auftreten. Sie dienen dazu, den Programmtext uebersichtlich und lesbar zu gestalten. An folgenden Stellen duerfen keine Leerzeichen stehen:

- vor der Zeilennummer einer Zeile
- innerhalb einer Zeilennummer

- innerhalb eines Schluesselwortes
- innerhalb eines Identifikators
- innerhalb einer numerischen Konstante
- innerhalb der zusammengesetzten Relationssymbole
  <> >< <= =< >= =>

Leerzeichen haben nur innerhalb von string-Konstanten (s. Abschn. 6.1.), offenen Zeichenreihen (s. Abschn. 10.1.1.) und Formatzeichenreihen (s. Abschn. 10.4.) eine Bedeutung.
In der Syntaxbeschreibung werden Leerzeichen nur dann explizit angegeben, wenn ihr Auftreten gefordert wird. Die Angabe erfolgt dann immer durch die metalinguistische Variable Leerzeichen.
Alle Zeichen, die zum ASCII-Zeichensatz, aber nicht zum BASIC-Zeichensatz gehoeren, duerfen im Quelltext eines Programms nicht auftreten (siehe jedoch Anleitung fuer den Bediener). Sie koennen jedoch in den Eingabezeichenreihen der Eingabeanweisung auftreten bzw. durch die Funktion CHR□ (s. Abschn. 6.4.) erzeugt werden.

3.2. Schluesselworte

Schluesselworte sind Identifikatoren, die in BASIC benutzt werden, um Anweisungen oder Teile von Anweisungen zu kennzeichnen.
BASIC verfuegt ueber folgende Schluesselworte:

ACCESS
AND
ANGLE
AREA
ARITHMETIC
ASK
AT
BACKGROUND
BASE
BEAM
BEGIN
BOUNDS
BREAK
CALL
CAPABILITY
CASE
CAUSE
CHAIN
CHARACTER
CLEAR
CLIP
CLOSE
COLLATE
COLOR
CONNECT
CONTINUE
COUNT
DATA
DATUM
DEBUG
DECIMAL
DECLARE
DEF
DEGREES
DELAY
DELETE
DEVICE
DIM
DISCONNECT
DISPLAY
DO
ELAPSED
ELSE
ELSEIF
END
ERASE
ERASABLE
EVENT
EXCEPTION
EXIT
EXLINE
EXTERNAL
EXTRACT
EXTYPE
FILETYPE
FILL
FIRST
FIXED
FOR
FROM
FUNCTION
GET
GO
GOSUB
GOTO
GRAPH
HANDLER
HEIGHT
IF
IMAGE
IN
INPUT
INTERNAL
IS
JUSTIFY
KEY
KEYED
LAST
LENGTH
LET
LINE
LINES
LIST
LOCATE
MARGIN
MAT
MESSAGE
MISSING
NAME
NATIVE
NEXT
NOT
NUMERIC
OF
OFF
ON
OPEN
OPTION
OR
ORGANIZATION
OUT
OUTIN

OUTPUT	RESET	STYLE
PARACT	REST	SUB
PARSTOP	RESTORE	TAB
PICTURE	RETRY	TEMPLATE
PLOT	RETURN	TEXT
POINT	REWRITE	THEN
POINTER	SAME	THERE
POINTS	SELECT	TIME
POSITION	SEND	TIMEOUT
PRINT	SEQUENTIAL	TO
PROCESS	SET	TRACE
PROGRAM	SETTER	UNTIL
PROMPT	SHARED	URGENCY
PUT	SIGNAL	USE
RADIANS	SIZE	USING
RANDOMIZE	SKIP	VARIABLE
READ	STANDARD	VIEWPORT
RECEIVE	START	WAIT
RECORD	STATUS	WHEN
RECSIZE	STEP	WHILE
RECTYPE	STOP	WINDOW
RELATIVE	STREAM	WITH
REM	STRING	WRITE
RENUMBER	STRUCTURE	ZONEWIDTH

Fuer die in Anlage 1 beschriebene Erweiterung des Standards werden ausserdem folgende Schluesselworte benutzt:

INTEGER PRELUDE

Zur Notation der Schluesselworte koennen wahlweise Gross- und Kleinbuchstaben verwendet werden.
Schluesselworte sind mit wenigen Ausnahmen (s. Abschn. 3.3.) nicht reserviert, d.h. ein Schluesselwort kann zur Bezeichnung eines Programmobjektes verwendet werden. Es haengt dann von der syntaktischen Position des Identifikators im Quelltext ab, ob er als Schluesselwort oder als Bezeichnung eines Programmobjektes fungiert.
Es wird jedoch empfohlen, Schluesselworte nicht als Bezeichnungen fuer Programmobjekte zu verwenden, da in anderen Standards, die an den ANSI-Standard angelehnt sind, alle Schluesselworte reserviert sind.
Vor und nach jedem Schluesselwort, das im Programm auftritt, muss ein Zeichen stehen, das weder Buchstabe, Ziffer noch _ oder □ ist. Auf das Schluesselwort kann auch unmittelbar das Zeilenendezeichen (EOL) folgen.

3.3. Reservierte Worte

Eine Reihe von Identifikatoren in BASIC sind reserviert, d.h. sie duerfen nur in der von BASIC festgelegten Bedeutung verwendet werden. Die Reservierung ist notwendig, um die eindeutige Analyse eines BASIC-Programms zu sichern.

Folgende Schluesselworte sind reserviert:

ELAPSED	INPUT	PLOT	REM	USING
ELSE	LINE	PRINT	REWRITE	WRITE
GET	LOCATE	PROMPT	SKIP	
GRAPH	NOT	READ	TIMEOUT	

Zu den reservierten Worten gehoeren weiterhin die Namen der folgenden, argumentlosen Standardfunktionen:

DATE EXTYPE PI TIME DATE□
EXLINE MAXNUM RND

Ausserdem sind die Namen der folgenden, speziellen Feldwerte reserviert.

CON IDN TRANSFORM ZER NUL□

3.4. Identifikatoren

Identifikatoren dienen als Bezeichnungen fuer Programmobjekte wie z.B. Variable, Felder, Funktionen, Unterprogramme, Programme usw.

Syntax

```
identifikator ::= < num_identifikator
                  | string_identifikator
                  | routine_identifikator >

num_identifikator ::= buchstabe [ buchstabe | ziffer | _ ] 30

string_identifikator ::= buchstabe [ buchstabe | ziffer | _ ] 29 □

routine_identifikator ::= buchstabe [ buchstabe | ziffer | _ ] 30
```

Ein Identifikator darf hoechstens aus 31 Zeichen bestehen. Unterscheiden sich zwei Identifikatoren nur in der Verwendung von Gross- und Kleinbuchstaben, so werden sie als gleich betrachtet. So werden z.B. die Identifikatoren Xa1, XA1 und xa1 als gleich betrachtet, waehrend X1 und X_1 unterschiedliche Identifikatoren sind.
Numerische Identifikatoren wie z.B.

X Summe letzter_Index MATRIX

dienen zur Bezeichnung numerischer Objekte (einfache numerische Variable, numerische Felder numerische Funktionen, numerische Feldwerte).

Analog dienen string-Identifikatoren wie z.B.

A□ Kopfzeile□ MONATSNAMEN□

zur Bezeichnung von string-Objekten (einfache string-Variable, string-Felder, string-Funktionen, string-Feldwerte).

Routine-Identifikatoren wie z.B.

Konvertierung sort Fehlerbehandlung

werden zur Bezeichnung von Unterprogrammen, Hauptprogrammmen und Fehlerbehandlungsroutinen verwendet. Da numerische und Routine-Identifikatoren syntaktisch die gleiche Form haben, haengt es von der Art des bezeichneten Objektes ab, was fuer ein Identifikator vorliegt.
Bei der Bildung von Identifikatoren ist zu beachten, dass die im Abschnitt 3.3. aufgefuehrten, reservierten Worte nicht verwendet werden duerfen.
Innerhalb eines Programms muessen alle Identifikatoren, die Programmeinheiten (Hauptprogramm, externe Subroutine, externe Funktion) bezeichnen, voneinander verschieden sein.
Innerhalb einer Programmeinheit darf ein Identifikator nur zur Bezeichnung eines Objektes (einfache Variable, Feld, Subroutine Funktion, selbststaendige Ausnahmeroutine) verwendet werden. Da diese Objekte lokal zur Programmeinheit gueltig sind, bezeichnet ein in zwei oder mehreren Programmeinheiten verwendeter Identifikator verschiedene Objekte.
Schluesselworte und die Namen der Standardfunktionen sind im gesamten Programm gueltig. Sie koennen jedoch innerhalb einer Programmeinheit zur Bezeichnung von Objekten verwendet werden, sofern sie nicht reserviert sind. In diesem Fall muessen sie jedoch explizit deklariert werden. Wird z.B. ein numerisches Feld mit dem Namen SIN deklariert, so bezeichnet der Identifikator SIN innerhalb der Programmeinheit dieses Feld und nicht die Standardfunktion. Die Standardfunktion SIN kann dann in dieser Programmeinheit nicht gerufen werden. Aus Gruenden der Lesbarkeit wird jedoch empfohlen, von solchen Moeglichkeiten nicht Gebrauch zu machen.

4. Grundlegende Programmstrukturen

Der vorliegende Abschnitt erlaeutert die generelle Strukturierung eines BASIC-Programms als Folge von Programmeinheiten bzw. Zeilen.

4.1. Programmeinheiten

Jedes BASIC-Programm besteht aus Programmeinheiten.

Syntax

```
programm ::= hauptprogramm [[kommentarzeile]* externe_prozedur]*

externe_prozedur ::= <| externe_funktionsdefinition |>
                     | externe_up_definition      |

programmeinheit ::= <| hauptprogramm |>
                    | externe_prozedur |
```

Die erste Programmeinheit eines BASIC-Programms muss ein Hauptprogramm sein. Dem Hauptprogramm koennen als weitere Programmeinheiten externe Prozeduren folgen. Zwischen den Programmeinheiten koennen wahlweise Kommentarzeilen stehen (s. Abschn. 4.2.). Solche Kommentarzeilen gehoeren zu keiner Programmeinheit und koennen deshalb niemals Ziel eines Sprunges sein.
Jede Programmeinheit bildet einen eigenen Gueltigkeitsbereich, d.h. alle Objekte, die in einer Programmeinheit definiert werden, sind nur in dieser Programmeinheit gueltig. Die Verbindung zwischen den Programmeinheiten erfolgt durch Rufe externer Prozeduren.

4.2. Zeilen, Kommentarzeilen

Jede Programmeinheit eines BASIC-Programms und damit das BASIC-Programm selbst, besteht aus einer Folge von Zeilen. Jede Zeile beginnt mit einer Zeilennummer, der sich im allgemeinen eine Anweisung anschliesst.

Syntax

```
zeilennummer ::= {ziffer}^5

zeilenrest ::= [!kommentar]eol

kommentar ::= [zeichen]*

zeilenfortsetzung ::= &zeilenrest&
```

kommentarzeile ::= zeilennummer <[rem_anweisung | !kommentar]>eol

rem_anweisung ::= REM kommentar

Der Wert einer Zeilennummer muss im Bereich von 1 bis 50000 liegen. Fuehrende Nullen sind zulaessig, jedoch duerfen insgesamt nur 5 Ziffern zur Bildung einer Zeilennummer verwendet werden. Die erste Ziffer der Zeilennummmer muss das erste Zeichen der Zeile sein.
Zeilennummern sind in zweierlei Hinsicht von Bedeutung:
- Sie legen eine statische Reihenfolge der Programmzeilen fest. Deshalb koennen bei der Eingabe bzw. bei der Editierung eines BASIC-Programms die Programmzeilen in beliebiger Reihenfolge eingegeben bzw. verarbeitet werden.
- Sie markieren jede Programmzeile. Zeilennummern werden deshalb im Programm als Marken verwendet, z.B. in goto- und gosub-Anweisungen. Derartige Referenzen duerfen nur innerhalb einer Programmeinheit auftreten.
  Weiterhin kann beim Editieren durch Angabe der Zeilennummer auf die entsprechende Zeile zugegriffen werden.

Auf die Zeilennummer erfolgt im allgemeinen eine Anweisung, z.B.

```
1570 LET A = B + 0.5
```

Jede Zeile wird durch einen Zeilenrest abgeschlossen (Ausnahmen sind die image-Zeile, s. Abschn. 10., und die Kommentarzeile mit rem-Anweisung, s.u.). Der wesentlichste Bestandteil des Zeilenrestes ist der Begrenzer eol (end of line), der das Ende der Zeile anzeigt. Die vorliegende Implementierung benutzt als eol die Zeichenkombination CR LF (carriage-return, line-feed). Der Begrenzer eol ist kein sichtbares Zeichen und wird deshalb in den Beispielen nicht ausgewiesen.
Vor dem Begrenzer eol kann ein Kommentar stehen. Dieser Kommentar wird durch ! eingeleitet und erstreckt sich bis zum eol.

```
380 LET A = A + 1    !A inkrementieren
```

Innerhalb des Kommentars koennen alle Zeichen des BASIC-Zeichensatzes verwendet werden.
Die auf die Zeilennummer folgende Anweisung kann auch ueber mehrere Zeilen notiert werden. Es ist deshalb zwischen einer logischen und einer physischen Zeile zu unterscheiden.
Als logische Zeile wird eine vollstaendige Programmzeile bezeichnet. Sie enthaelt neben der Zeilennummer die vollstaendige Anweisung und kann sich ueber eine oder mehrere physische Zeilen erstrecken.
Als physische Zeile werden die Zeichen zwischen zwei aufeinander folgenden Begrenzern eol bezeichnet. Eine physische Zeile darf maximal 132 Zeichen enthalten.
Soll sich eine logische Zeile ueber mehrere physische Zeilen erstrecken, so ist zwischen diesen physischen Zeilen jeweils eine Zeilenfortsetzung anzugeben. So kann z.B. die logische Zeile

```
100 FUNCTION MITTELWERT (MESSWERTE (),N)
```

auch als

```
100 FUNCTION MITTELWERT        &
&            (MESSWERTE (),N)
```

oder

```
100 FUNCTION MITTELWERT    ! arithmetisches Mittel                &
&            (Messwerte(),! eindimensionales Feld mit Messwerten&
&            N)           ! Anzahl der Messwerte
```

notiert werden. Zu beachten ist, dass das Zeichen &, sofern es den Beginn einer physischen Zeile kennzeichnet, stets das erste Zeichen der Zeile sein muss. Die Trennung einer logischen Zeile kann an allen Stellen erfolgen, an denen ein Leerzeichen stehen darf. Ausgenommen davon sind Leerzeichen in string-Konstanten, offenen Zeichenreihen, Formatzeichenreihen und Kommentaren.
Fuer die Kommentierung eines BASIC-Programms stehen zwei Moeglichkeiten zur Verfuegung:
- Angabe eines Kommentars im Anschluss an die Anweisung durch einen entsprechenden Zeilenrest (s. o.) und
- Angabe gesonderter Kommentarzeilen.

Kommentarzeilen wie z. B.

```
100 REM
110 REM EROEFFNEN DER DATEIEN
120 REM
```

oder

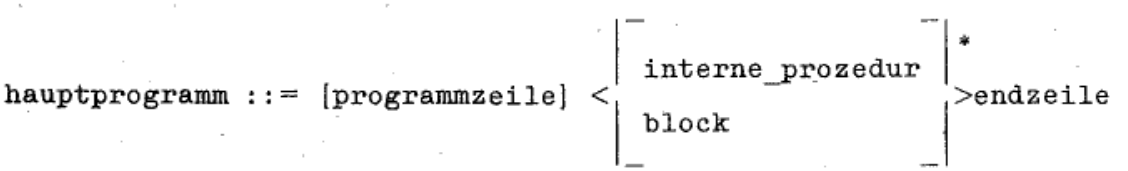
```
200 !FEHLERKONTROLLE
```

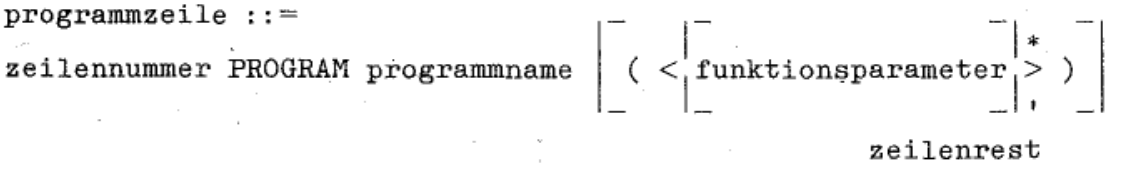
duerfen sich jeweils nur ueber eine physische Zeile erstrecken. Wird bei der Programmabarbeitung eine Kommentarzeile erreicht, so erfolgt ohne Wirkung ein Uebergang auf die folgende Zeile. Kommentarzeilen koennen in einem BASIC-Programm an beliebigen Stellen auftreten.

4. 3. Struktur des Hauptprogramms

Jedes BASIC-Programm enthaelt als erste Programmeinheit ein Hauptprogramm. Der folgende Abschnitt beschreibt dessen grundlegende Struktur. Die anderen Programmeinheiten (externe Funktionen und Unterprogramme) werden im Abschnitt 9. beschrieben.

Syntax





```
programmname ::= routine_identifikator

                          ┌                                  ┐
                          │ interne_funktionsdefinition      │
interne_prozedur ::= <    │ interne_up_definition            │>
                          │ fehlerbehandlungsroutine         │
                          └                                  ┘

              ┌                        ┐
              │ anweisungszeile        │
              │ schleife               │
              │ if_block               │
block ::= <   │                        │>
              │ select_block           │
              │ image_zeile            │
              │ protection_block       │
              └                        ┘

anweisungszeile ::= zeilennummer anweisung zeilenrest

                   ┌                        ┐
                   │ deklarative_anweisung  │
anweisung ::= <    │ imperative_anweisung   │>
                   │ bedingte_anweisung     │
                   └                        ┘

endzeile ::= zeilennummer end_anweisung zeilenrest

end_anweisung ::= END

stop_anweisung ::= STOP
```

Jedes Hauptprogramm kann mit einer Programmzeile beginnen, z.B.

100 PROGRAM ABRECHNUNG

Im Interesse einer guten Strukturierung des Programms sollte immer eine Programmzeile angegeben werden, obwohl dies nicht notwendig ist.
Fuer ein Programm, das vom Bediener gestartet wird, hat eine Programmzeile keinerlei Wirkung. Bei einem solchen Programm duerfen in der Programmzeile keine Parameter auftreten.
Fuer ein Programm, das von einem anderen Programm ueber die chain-Anweisung (s. Abschn. 9.) gestartet wird, koennen ueber die Parameter der Programmzeile Werte aus dem rufenden in das gerufene Programm uebergeben werden.

Mit Hilfe der chain-Anweisung kann auch ein Programm gerufen werden, das keine Programmzeile enthaelt. Diesem Programm koennen keine Werte uebergeben werden.
Der Gueltigkeitsbereich der Parameter in der Programmzeile ist das Hauptprogramm. Sie duerfen im Hauptprogramm nicht explizit deklariert werden.
Auf die Programmzeile folgen im Hauptprogramm Definitionen interner Prozeduren und sogenannte Bloecke. Unter einem Block werden Anweisungszeilen sowie verschiedene zusammengehoerende Gruppen von Zeilen verstanden.
Die in Anweisungszeilen auftretenden Anweisungen werden in deklarative, imperative und bedingte Anweisungen eingeteilt. Durch

deklarative Anweisungen werden Programmobjekte definiert bzw. spezifiziert und bestimmte Festlegungen fuer die Programmabarbeitung getroffen. Zu den deklarativen Anweisungen gehoeren:

data-Anweisung
declare-Anweisung
dimension-Anweisung
option-Anweisung

Imperative Anweisungen fuehren Aktionen aus. Dazu gehoeren u. a. solche Anweisungen wie:

call-Anweisung
close-Anweisung
goto-Anweisung
input-Anweisung
let-Anweisung
open-Anweisung
return-Anweisung
stop-Anweisung
write-Anweisung

und viele andere.

Bedingte Anweisungen fuehren in Abhaengigkeit von Bedingungen verschiedene Aktionen aus. Bedingte Anweisungen sind:

if-Anweisung
on-gosub-Anweisung
on-goto-Anweisung

Das Hauptprogramm wird durch eine end-Zeile abgeschlossen.
Wird bei der Programmabarbeitung die end-Zeile erreicht, so wird die Programmabarbeitung abgebrochen. Der Abbruch der Programmabarbeitung kann weiterhin auch durch eine stop-Anweisung erreicht werden. Ein Programm kann mehrere stop-Anweisungen in verschiedenen Programmeinheiten enthalten.
Die Abarbeitung eines Programms beginnt mit der ersten Zeile. Die Zeilen werden in sequentieller Reihenfolge solange abgearbeitet, bis

- die Abarbeitung einer Anweisung (z.B. die goto-Anweisung) diese sequentielle Folge unterbricht,
- eine Ausnahme ausgeloest wird (falls es nicht eine triviale Ausnahme ist, die durch die Standardfehlerbehandlung ausgewertet wird),
- eine chain-Anweisung abgearbeitet wird oder
- eine end- bzw. stop-Anweisung abgearbeitet wird.

5. Verarbeitung numerischer Werte

Numerische Werte bilden einen der zwei primitiven Datentypen von BASIC. Sie koennen durch Konstanten dargestellt und in Variablen gespeichert werden.
Numerische Werte sind die Approximation (naeherungsweise Darstellung) gebrochener (reeller) Zahlen. Die naeherungsweise Darstellung resultiert aus der begrenzten Speicherkapazitaet und der rechnerinternen Darstellung fuer einen numerischen Wert.
Die durch die Approximation gewonnene Genauigkeit betraegt 6 bzw. 16 Dezimalstellen (s. Abschn. 5.6.), d.h. nur die ersten 6 bzw. 16 signifikanten Dezimalstellen einer reellen Zahl werden durch einen numerischen Wert exakt dargestellt, waehrend in weiteren Dezimalstellen Ungenauigkeiten auftreten.
Numerische Werte koennen positiv, negativ oder Null sein.
Ein numerischer Wert W, der nicht Null ist, muss im Bereich

$$1.2 * 10^{-38} <= |W| <= 3.4 * 10^{+38}$$

bei numerischen Groessen einfacher Genauigkeit bzw. im Bereich

$$2.3 * 10^{-308} <= |W| <= 1.7 * 10^{+308}$$

bei numerischen Groessen doppelter Genauigkeit liegen. Der groesste numerische Wert (in Interndarstellung) wird von der Standardfunktion MAXNUM (s. Abschn. 5.4.) bereitgestellt. Wird bei der Programmabarbeitung ein numerischer Wert berechnet, dessen Betrag groesser als MAXNUM ist, so wird eine nicht triviale Ausnahme (Ueberlauf) ausgeloest. Wird ein Wert berechnet, dessen Betrag zwischen Null und 10^{-38} bzw. 10^{-308} liegt, so wird eine triviale Ausnahme (Unterlauf) ausgeloest und als Wert fuer folgende Rechnungen Null angenommen.

5.1. Numerische Konstanten

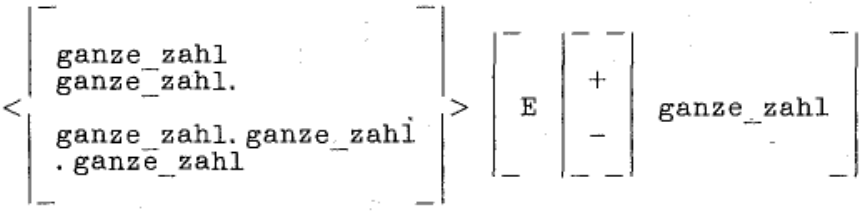
Eine numerische Konstante bezeichnet einen festen numerischen Wert in Dezimaldarstellung. Dieser Wert ist die Zahl, die durch die Konstante repraesentiert wird.

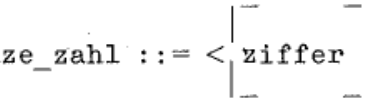
Syntax

konstante ::= num_konstante

num_konstante ::= [+ | -] num_konstante_ohne_vorzeichen

num_konstante_ohne_vorzeichen ::=





Ist kein Vorzeichen angegeben bzw. folgt auf "E" kein Vorzeichen (+ oder -), so wird ein positives Vorzeichen (+) angenommen.
Numerische Konstanten koennen auf verschiedene Arten dargestellt werden:

- als ganze Zahl ohne Exponent
  Beispiel: +38 -1404 413

  (Bemerkung: Derartige ganze Zahlen werden in BASIC immer als reele Zahlen betrachtet und intern entsprechend dargestellt, s. Anl. 1).

- als gebrochene Zahlen ohne Exponent
  Beispiel: 13. +13.0 -24.501 .403

- als ganze Zahlen mit Exponent
  Beispiel: 1E+10 1E10 -106E-03 -106e-3

- als gebrochene Zahlen mit Exponent
  Beispiel: +56.38E-14 -0.6E-12 6.e-1 .6e0

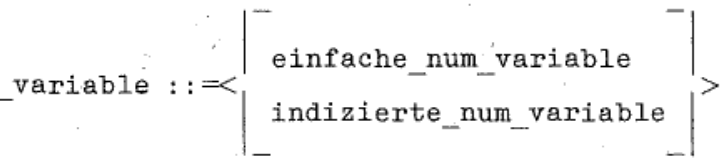
Konstanten bezeichnen die Werte in Dezimaldarstellung. Bei der Notation von Konstanten koennen beliebig viele Ziffernstellen angegeben werden. Da die Konstanten zur Programmabarbeitung jedoch in ihre interne Form konvertiert werden muessen, sind die eingangs erwaehnten Erlaeuterungen ueber den Wertebereich und die Darstellungsgenauigkeit numerischer Werte zu beachten.

5.2. Numerische Variable

Numerische Variable sind Groessen, die einen einzelnen, numerischen Wert enthalten koennen.

Syntax

variable ::= num_variable



einfache_num_variable ::= num_identifikator

indizierte_num_variable ::= num_feld (<index[,]>)

num_feld ::= num_identifikator

index ::= num_ausdruck

einfache_variable ::= einfache_num_variable

feld ::= num_feld

Zu jedem Zeitpunkt der Programmabarbeitung besitzt jede numerische Variable einen numerischen Wert. Dieser Wert kann durch Abarbeitung von Anweisungen veraendert werden.
Einfache numerische Variable wie z.B.

```
X     summe     I2
```

werden entweder implizit durch ihr Auftreten im Programm oder explizit in einer declare-Anweisung deklariert.
Der Gueltigkeitsbereich einer numerischen Variablen ist die Programmeinheit, es sei denn, dass diese Variable als Parameter einer internen Funktion auftritt.
Ein Index ist ein numerischer Ausdruck, dessen Wert gerundet wird. Der gerundete Wert von X ist durch INT(X+0.5) definiert (s. Abschn. 5.4.).
Indizierte numerische Variable wie z.B.

```
V(4)      table(i, j+1)
```

sind Elemente von Feldern. Der numerische Identifikator gibt an, zu welchem Feld die indizierte Variable gehoert. Durch die Indizes wird genau ein Element aus diesem Feld ausgewaehlt. Der zulaessige Bereich, den jeder Index annehmen darf, wird bei der Deklaration des Feldes in einer dimension- oder declare-Anweisung (s. Abschn. 7.1.) festgelegt. Liegt ein Index ausserhalb dieses Bereiches, so wird eine nicht triviale Ausnahme ausgeloest. Die Anzahl der Indizes einer indizierten Variablen muss mit der Anzahl der Dimensionen des Feldes uebereinstimmen.
Bei Betreten einer Programmeinheit besitzen saemtliche in ihr definierten numerischen Variablen den Wert Null.

5.3. Numerische Ausdruecke

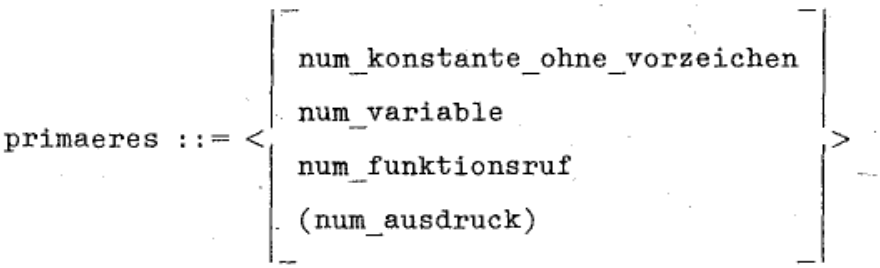
Numerische Ausdruecke werden aus numerischen Operanden (Konstanten, Variablen, Funktionsrufe) und den Operatoren fuer Addition, Subtraktion, Multiplikation, Division und Potenzierung gebildet. Ihre Berechnung liefert jeweils einen einfachen numerischen Wert.

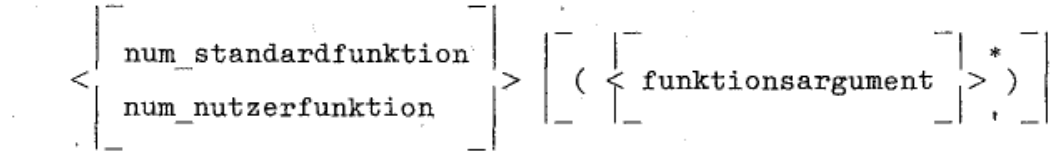
Syntax

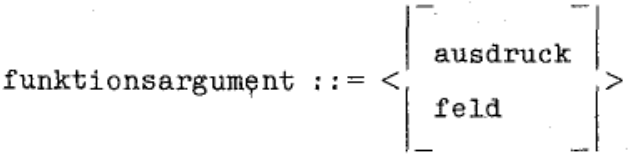
ausdruck ::= num_ausdruck

num_ausdruck ::= [+ | -] term [< [+ | -] > term]*

faktor ::= primaeres [^ primaeres]*







Die Notation und Berechnung numerischer Ausdruecke folgt den normalen Regeln der Algebra. Die Operatoren haben die uebliche Bedeutung:

+	Addition bzw. einseitiges (monadisches) Plus
-	Subtraktion bzw. einseitiges (monadisches) Minus
*	Multiplikation
/	Division
^	Potenzierung

Die Abarbeitungsreihenfolge der Operatoren wird durch Prioritaeten festgelegt.

^	hoechste Prioritaet
* /	
+ - (dyadisch und monadisch)	niedrigste Prioritaet

Operatoren mit hoeherer Prioritaet werden vor Operatoren mit niederer Prioritaet abgearbeitet. Folgen mehrere Operatoren gleicher Prioritaet aufeinander, so erfolgt die Abarbeitung von links nach rechts, falls nicht unter Ausnutzung der Kommutativitaet bzw. Assoziativitaet von Operatoren eine Umordnung der Operatoren bzw. eine andere Berechnungsreihenfolge festgelegt wird, um eine optimale Abarbeitung zu erzielen.
Durch die Verwendung von Klammern kann die Abarbeitungsreihenfolge anders festgelegt bzw. eine Umordnung der Operatoren verhindert werden. Ist ein Operand einer Operation ein geklammerter Ausdruck, so wird zunaechst der Wert des geklammerten Ausdrucks berechnet und anschliessend die Operation ausgefuehrt.

In den folgenden Beispielen wird durch vollstaendige Klammerung die Abarbeitungsreihenfolge verdeutlicht.

Ausdruck	Abarbeitung ohne Umordnung der Operanden	Abarbeitung mit Umordnung der Operanden
A * B + C	(A * B) + C	(B * A) + C
A + B * C	A + (B * C)	(B * C) + A
A + B * C ^ D	A + (B * (C ^ D))	A + ((C ^ D) * B)
A + B - C	(A + B) - C	(B + A) - C
A * B/C * D	((A * B)/C) * D	(B * A)/(C * D)
A ^ B ^ C	(A ^ B) ^ C	
- A * B	-(A * B)	-(B * A)

Der Ausdruck 0^0 liefert den Wert 1.
Folgende Faelle fuehren neben Ueberlaeufen bei der Berechnung numerischer Ausdruecke zu nicht trivialen Ausnahmen:
- Es wird eine Division durch Null versucht.
- Die Basis einer Potenzierung (1. Operand) ist Null und der Exponent (2. Operand) ist negativ.
- Die Basis einer Potenzierung ist negativ und der Exponent ist keine ganze Zahl.

Der Aufruf einer numerischen Funktion bewirkt die Unterbrechung der Berechnung des Ausdrucks, die Bereitstellung der Funktionsargumente und anschliessend die Abarbeitung der Funktion (s. Abschn. 9.). Nach der Abarbeitung der Funktion identifiziert die Funktionsbezeichnung einen einfachen numerischen Wert, mit dem die Berechnung des Ausdrucks fortgesetzt wird.
Die Anzahl und die Typen der Argumente in einem Funktionsruf muessen mit der Anzahl und den Typen der korrespondierenden formalen Parameter in der Funktionsdefinition uebereinstimmen. Ist als Funktionsargument ein Feld angegeben, so muss es die gleiche Zahl von Dimensionen wie der korrespondierende Parameter haben. Die Parametervermittlung erfolgt nach dem Prinzip "call by value" (s. Abschn. 9.).
Wird eine externe numerische Funktion gerufen, so muss die Arithmetik-Option dieser Funktion mit der Arithmetik-Option der rufenden Programmeinheit uebereinstimmen.
Jede in einem Ausdruck gerufene Nutzerfunktion muss in der gleichen Programmeinheit zuvor als Nutzerfunktion ausgewiesen werden. Dies kann entweder durch die Definition der Funktion (interne Funktionsdefinition, s. Abschn. 9.) oder durch Angabe der Funktion in einer declare-Anweisung erfolgen. Die Angabe in einer declare-Anweisung ist notwendig, wenn die gerufene Funktion extern, d.h. eine nicht in der vorliegenden Programmeinheit definierte Funktion ist bzw. wenn eine interne Funktion vor ihrer Definition gerufen werden soll.
Standardfunktionen sind a priori bekannt und koennen ohne besondere Vorkehrungen gerufen werden.

5.4. Numerische Standardfunktionen

Eine Reihe wichtiger mathematischer Funktionen und Algorithmen werden als Standardfunktionen bereitgestellt. Neben den in diesem Abschnitt vorgestellten Standardfunktionen existieren weitere (s. Abschn. 6., 7. und 12.).

Syntax

num_standardfunktion ::=

```
ABS | ACOS | ANGLE | ASIN | ATN | CEIL | COS | COSH | COT |
CSC | DATE | DEG | EPS | EXP | FP | MAXNUM | INT | IP | LOG |
LOG10 | LOG2 | MAX | MIN | MOD | PI | RAD | REMAINDER |
RND | ROUND | SEC | SGN | SIN | SINH | SQR |
TAN | TANH | TIME | TRUNCATE
```

randomize_anweisung ::= RANDOMIZE

In der folgenden Tabelle werden die oben aufgefuehrten Standardfunktionen naeher erlaeutert. Dabei stehen X und Y jeweils fuer numerische Ausdruecke und I fuer einen Index, d.h. einen Ausdruck, der einen ganzzahligen Wert liefert.
Alle Argumente der Funktionen koennen beliebige Werte annehmen, sofern dies nicht anders vermerkt ist. Bei Funktionen, die einen Winkel als Ergebnis liefern bzw. die einen Winkel als Argument besitzen, ist es von der Winkel-Option (s. Abschn. 5.6.) abhaengig, ob der Winkel im Grad- oder Bogenmass dargestellt wird.
Die Zeichenkette "pi" wird anstelle der Konstanten 3.14159... verwendet.

Tabelle 1: Standardfunktionen

Funktion	Funktionswert
ABS(X)	Betrag von X
ACOS(X)	Arcuscosinus von X im Bogen- oder Gradmass, wobei 0 <= ACOS(X) <= pi gilt; X muss im Bereich -1 <= X <= 1 liegen
ANGLE(X,Y)	Winkel im Grad- oder Bogenmass zwischen der positiven Abzisse und dem Vektor, der vom Koordinatenursprung zum Punkt mit den Koordinaten (X,Y) zeigt, wobei -pi < ANGLE(X,Y) <= pi gilt; X und Y duerfen nicht beide Null sein; die Winkelmessung erfolgt mathematisch positiv (entgegen dem Uhrzeiger), d.h. ANGLE(1,1) = 45 Grad
ASIN(X)	Arcussinus von X im Bogen- oder Gradmass, wobei -pi/2 <= ASIN(X) <= pi/2 gilt; X muss im Bereich -1 <= X <= 1 liegen

Tabelle 1: Fortsetzung

1	2
ATN(X)	Arcustangens von X im Bogen- oder Gradmass, wobei -pi/2 < ATN(X) < pi/2 gilt
CEIL(X)	kleinste ganze Zahl, die nicht kleiner als X ist; z.B. CEIL(1.3) = 2 und CEIL(-1.3.) = -1
COS(X)	Cosinus von X (X im Bogen- oder Gradmass)
COSH(X)	Hyperbelcosinus von X
COT(X)	Cotangens von X (X im Bogen- oder Gradmass)
CSC(X)	Cosekans von X (X im Bogen- oder Gradmass)
DATE	aktuelles Datum als Dezimalzahl in der Form JJTTT, wobei JJ die letzten zwei Ziffern des Jahres und TTT die Ordnungszahl des Tages im Jahr ist; der Wert von DATE fuer den 9. Mai 1977 ist folglich 77129
DEG(X)	Winkel im Gradmass, entsprechend dem Winkel X im Bogenmass
EPS(X)	Maximum von (X-X',X"-X, sigma), wobei X' und X" der Vorgaenger und der Nachfolger von X und sigma die kleinste darstellbare Zahl ist; EPS (0) liefert sigma; EPS kann fuer veschiedene Arithmetik-Options auch verschiedene Resultate liefern
EXP(X)	Exponentialwert von X, d.h. der Wert von e (e = Basis des natuerlichen Logarithmus) hoch X
FP(X)	gebrochener Teil von X, d.h. X-IP(X)
INT(X)	groesste ganze Zahl, die nicht groesser als X ist; z.B. INT(1.3) = 1 und INT(-1.3.) = -2
IP(X)	ganzzahliger Teil von X, d.h. SGN(X)*INT(ABS(X))
LOG(X)	natuerlicher Logarithmus von X; X muss groesser als Null sein
LOG10(X)	dekadischer Logarithmus von X; X muss groesser als Null sein
LOG2(x)	dyadischer Logarithmus von X; X muss groesser als Null sein
MAX(X,Y)	Maximum von X und Y

Tabelle 1: Fortsetzung

1	2
MAXNUM	groesster darstellbarer positiver Wert; MAXNUM liefert bei verschiedenen Arithmetik-Options unterschiedliche Werte
MIN(X,Y)	Minimum von X und Y
MOD(X,Y)	X modulo Y, d.h. X-Y*INT(X/Y); Y darf nicht Null sein
PI	Wert der Konstanten pi
REMAINDER(X,Y)	Divisionsrest von X/Y, d.h. X-Y*IP(X/Y); Y darf nicht Null sein
RND	naechste Zufallszahl aus der Folge von Zufallszahlen, wobei 0 <= RND < 1 gilt
ROUND(X,N)	Wert von X, gerundet auf N Dezimalstellen rechts vom Dezimalpunkt (N > 0) bzw. N Dezimalstellen links vom Dezimalpunkt (N < 0), d.h. INT(X*10^N+0.5)/10^N
SEC(X)	Sekans von X (X im Bogen- oder Gradmass)
SGN(X)	Vorzeichen von X: -1 falls X < 0 0 falls X = 0 1 falls X > 0
SIN(X)	Sinus von X (X im Bogen- oder Gradmass)
SINH(X)	Hyperbelsinus von X
SQR(X)	nicht-negative Quadratwurzel von X; X darf nicht negativ sein
TAN(X)	Tangens von X (X im Bogen- oder Gradmass)
TANH(X)	Hyperbeltangens von X
TIME	Zeit in Sekunden, seit der letzten Mitternacht; fuer 11.15 liefert TIME 40500; der Wert von TIME fuer Mitternacht ist Null
TRUNCATE(X,N)	Wert von X, abgeschnitten auf N Dezimalstellen rechts von Dezimalpunkt (N > 0) bzw. N Dezimalstellen links vom Dezimalpunkt (N < 0) d.h. IP(X*10^N)/10^N

Folgende Faelle fuehren neben Ueberlaufen bei der Berechnung der Funktionswerte zu nicht trivialen Ausnahmen:

- Das Argument von LOG, LOG10 oder LOG2 ist Null oder negativ.
- Das Argument von SQR ist negativ.
- Das zweite Argument von MOD oder REMAINDER ist Null.
- Das Argument von ACOS oder ASIN ist kleiner als -1 oder groesser als 1.
- Beide Argumente von ANGLE sind Null.

Das BASIC-Laufzeitsystem enthaelt einen Generator fuer Pseudo-Zufallszahlen, der bei jedem Aufruf von RND eine neue Zufallszahl liefert. Solange keine randomize-Anweisung verwendet wird, liefert der Generator bei jedem Programmlauf die gleiche Folge von Zufallszahlen. Die randomize-Anweisung aendert diese Folge, so dass bei verschiedenen Programmlaeufen auch verschiedene Folgen von Zufallszahlen erzeugt werden.
Der Zufallszahlengenerator ist global fuer das gesamte Programm.

5.5. Let-Anweisung fuer numerische Werte

Die let-Anweisung fuer numerische Werte gestattet die Zuweisung eines durch einen numerischen Ausdruck berechneten numerischen Wertes an eine oder mehrere numerische Variable.

Syntax

```
let-anweisung ::= num_let_anweisung

num_let_anweisung ::=

        ┌                 ┐*
     LET <│ num_variable  │>  = num_ausdruck
        └                 ┘;
```

Den Variablen links vom Gleichheitszeichen (Zielvariable) wird der Wert des Ausdruckes rechts vom Gleichheitszeichen zugewiesen. Diesen Wert behalten die Zielvariablen solange, bis ihnen ein neuer Wert zugewiesen wird.
Das Schluesselwort LET muss im Gegensatz zu anderen BASIC-Versionen stets angegeben werden.
Die Abarbeitung der numerischen let-Anweisung erfolgt in folgenden Schritten:
- Sind indizierte Zielvariablen vorhanden, so werden deren Indizes in der Reihenfolge von links nach rechts berechnet.
- Der Wert des numerischen Ausdruckes rechts vom Gleichheitszeichen wird berechnet.
- Der berechnete Wert wird den Zielvariablen in der Reihenfolge von links nach rechts zugewiesen.

In der Anweisung

```
LET A(I+1), B,C(J) = (Y+Z)*3.25
```

werden zunaechst die Indizes I+1 und J berechnet und anschliessend der Ausdruck (Y+Z)*3.25. Dann wird den Variablen A(I+1), B und C(J) der berechnete Wert zugewiesen.
Zu beachten ist, dass die Anweisungen

```
LET I=1
LET I, A(I)=2
```

nicht aequivalent zu

```
LET I=1
LET I=2
LET A(I)=2
```

sind, sondern den Anweisungen

```
LET I=1
LET I=2
LET A(1)=2
```

entsprechen.

5.6. Numerische Genauigkeit, Winkeldarstellung und Deklaration von numerischen Variablen

Numerische Werte werden normalerweise als dezimale Gleitkommazahlen mit einer bestimmten Genauigkeit aufgefasst. Durch eine option-Anweisung kann diese Genauigkeit festgelegt werden.
Alle im Zusammenhang mit den trigonometrischen Standardfunktionen auftretenden Winkel koennen im Bogen- oder Gradmass dargestellt werden. Dies kann ebenfalls durch eine option-Anweisung festgelegt werden.
Die declare-Anweisung erlaubt die explizite Deklaration von numerischen Variablen.

Syntax

```
                                  [        ]*
option_anweisung ::= OPTION <     [ option ]>
                                  [        ],

                 [                [        ]   ]
                 [                [ DECIMAL ]   ]
                 [ ARITHMETIC <   [        ]>  ]
                 [                [ NATIVE  ]   ]
option ::= <     [                [        ]   ]>
                 [        [        ]          ]
                 [ ANGLE <  [ DEGREES ]>     ]
                 [        [ RADIANS ]          ]
                 [        [        ]          ]

declare_anweisung ::= DECLARE typ-deklaration

                                  [                        ]*
typ_deklaration ::= NUMERIC <     [ einfache_num_variable  ]>
                                  [                        ],
```

Mit einer Arithmetik-Option kann festgelegt werden, mit welcher Genauigkeit numerische Werte in einer Programmeinheit dargestellt werden. Pro Programmeinheit darf hoechstens eine Arithmetik-Option angegeben werden. Sie muss angegeben werden, bevor ein numerischer Ausdruck oder eine dim- bzw. declare-Anweisung auftritt.

Wenn keine Arithmetik-Option oder die Anweisung

```
OPTION ARITHMETIC DECIMAL
```

angegeben wird, so sind alle numerischen Werte dezimale Gleitkommazahlen mit 16 Stellen Genauigkeit (doppelte Genauigkeit). Alle numerischen Variablen belegen dann im Speicher 8 Byte.
Wird die Anweisung

```
OPTION ARITHMETIC NATIVE
```

angegeben, so sind alle numerischen Werte dezimale Gleitkommazahlen mit 6 Stellen Genauigkeit. Alle numerischen Variablen belegen dann im Speicher 4 Byte (einfache Genauigkeit).
Eine Arithmetik-Option gilt innerhalb einer Programmeinheit. Deshalb koennen innerhalb einer Programmeinheit niemals numerische Groessen einfacher und doppelter Genauigkeit gleichzeitig verarbeitet werden. Innerhalb eines Programms koennen verschiedene Programmeinheiten allerdings mit verschiedenen Arithmetik-Options versehen werden. Der Datenaustausch zwischen solchen Programmeinheiten kann dann allerdings nur ueber einfache string-Werte erfolgen.
Mit einer Winkel-Option kann festgelegt werden, wie in einer Programmeinheit die Winkelangaben fuer die trigonometrischen Funktionen dargestellt werden. Pro Programmeinheit darf hoechstens eine Winkel-Option angegeben werden. Sie muss angegeben werden, bevor eine numerische Standardfunktion gerufen wird.
Wenn keine Winkel-Option oder die Anweisung

```
OPTION ANGLE RADIANS
```

angegeben wird, so verwenden die numerischen Standardfunktionen COS, COT, CSC, SEC, SIN und TAN Argumente im Bogenmass bzw. liefern die numerischen Standardfunktionen ACOS, ANGLE, ASIN und ATN Resultate im Bogenmass.
Wird die Anweisung

```
OPTION ANGLE DEGREES
```

angegeben, so sind die Argumente bzw. Resultate der oben aufgefuehrten Funktionen Angaben im Gradmass.
Wird bei der Programmabarbeitung eine Zeile erreicht, die eine option-Anweisung enthaelt, so wird diese Anweisung ohne Wirkung uebergangen.
Mit einer declare-Anweisung koennen einfache numerische Variable und numerische Felder deklariert werden. Die Deklaration numerischer Felder wird in Abschnitt 7. erlaeutert.
Durch

```
DECLARE NUMERIC X, Y
```

wird festgelegt, dass X und Y einfache numerische Variable sind. Diese Anweisung ist allerdings nicht notwendig, da einfache numerische Variable auch implizit durch ihr Auftreten im Programm deklariert werden koennen.
Eine numerische Variable darf hoechstens einmal in einer declare-Anweisung deklariert werden. Parameter von Funktionen und Prozeduren duerfen nicht in einer declare-Anweisung auftreten.
Wird bei der Programmabarbeitung eine Zeile erreicht, die eine declare-Anweisung enthaelt, so wird diese Anweisung ohne Wirkung uebergangen.

6. Verarbeitung von String-Werten

String-Werte bzw. Zeichenketten-Werte bilden einen der zwei primitiven Datentypen von BASIC. Sie koennen durch Konstanten dargestellt und in Variablen gespeichert werden.
String-Werte sind beliebige Folgen von Zeichen, wobei die Anzahl der Zeichen zwischen 0 und 32767 liegen kann. Bei einer String-Variablen kann festgelegt werden, wieviele Zeichen ein String-Wert maximal haben darf, der in dieser String-Variablen gespeichert werden soll.

6.1. String-Konstanten

Eine String-Konstante bezeichnet einen festen String-Wert, d.h. sie stellt eine Zeichenreihe fester Laenge dar, die von Anfuehrungszeichen eingeschlossen wird.

Syntax

```
konstante ::= string_konstante

string_konstante ::= " [zeichen]* "
```

Die Laenge einer String-Konstanten, d.h. die Zahl der zwischen den Anfuehrungszeichen stehenden Zeichen wird durch die Laenge einer physischen Zeile begrenzt und kann maximal 129 betragen. Als Zeichen sind Buchstaben, Ziffern und die Sonderzeichen 1 (s. Abschn. 3.1.) zulaessig. Ein Anfuehrungszeichen innerhalb einer String-Konstanten muss durch zwei unmittelbar aufeinander folgende Anfuehrungszeichen dargestellt werden. Diese zwei Anfuehrungszeichen zaehlen jedoch nur als ein Zeichen.
Leerzeichen innerhalb von String-Konstanten sind signifikant. Gross- und Kleinbuchstaben werden unterschieden.
In den folgenden Beispielen wird fuer einige String-Werte ihre Notation als String-Konstante dargestellt.

String-Wert	String-Konstante	
XYZ	"XYZ"	\ diese String-Werte
XyZ	"XyZ"	/ sind verschieden
FEHLERSTATUS = "HELLO"	"FEHLERSTATUS = """HELLO"""	dieser String-Wert enthaelt 7 Zeichen

Ein leerer String-Wert (String-Wert mit der Laenge 0) kann durch die String-Konstante "" dargestellt werden.

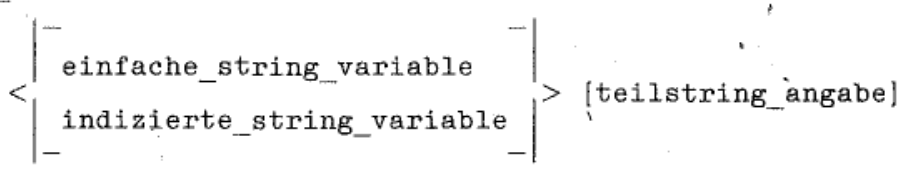
6.2. String-Variable

String-Variable sind Groessen, die einen einzelnen String-Wert enthalten koennen. String-Variable werden von numerischen Variablen syntaktisch dadurch unterschieden, dass ihre Bezeichnung stets mit dem Zeichen ¤ endet.

Syntax
------

variable ::= string_variable

string_variable ::=



einfache_string_variable ::= string_identifikator

indizierte_string_variable ::= string_feld ({index}^{3},)

string_feld ::= string_identifikator

teilstring_angabe ::= (index : index)

einfache_variable ::= einfache_string_variable

feld ::= string_feld

Zu jedem Zeitpunkt der Programmabarbeitung besitzt jede String-Variable einen String-Wert. Dieser Wert kann durch Abarbeitung von Anweisungen geaendert werden. Die Laenge dieses String-Wertes, d.h. die Anzahl der Zeichen, kann zwischen Null und einem fuer die String-Variable definierten Maximum liegen (s. Abschn 6.6.).
Einfache String-Variable wie z.B.

```
Name□      A□     Ueberschrift□
```

werden entweder implizit durch ihr Auftreten im Programm oder explizit in einer declare-Anweisung (s. Abschn. 6.6.) deklariert. Der Gueltigkeitsbereich einer String-Variablen ist die Programmeinheit, es sei denn, dass diese Variable als Parameter in einer internen Funktion auftritt.
Indizierte String-Variable wie z.B.

```
tabelle□(15)      MATRIX□(I,I+1)
```

sind Elemente von Feldern. Der String-Identifikator gibt an, zu welchem Feld die indizierte Variable gehoert. Durch die Indizes wird genau ein Element aus diesem Feld ausgewaehlt. Der zulaessige Bereich, den jeder Index annehmen darf, wird bei der Deklaration des Feldes in einer dimension- oder declare-Anweisung (s. Abschn. 7.1.) festgelegt. Liegt ein Index ausserhalb dieses Bereiches, so wird eine nicht triviale Ausnahme ausgeloest. Die Anzahl der Indizes einer indizierten Variablen muss mit der Anzahl den Dimensionen des Feldes uebereinstimmen.
Durch eine Teilstring-Angabe wird ein String-Wert spezifiziert, der einen Teil des String-Wertes darstellt, der der String-Variablen zugeordnet ist. So wird durch

```
A□(m:n)
```

ein String-Wert spezifiziert, der aus dem m-ten bis n-ten Zeichen des String-Wertes besteht, der der einfachen String-Variablen A□ zugeordnet ist. Dabei muessen m und n Indizes, d.h. ganzzahlige Werte sein. Gegebenenfalls wird gerundet. Die Zeichen in einem String-Wert werden von links nach rechts, beginnend mit 1 durchnumeriert. Falls also A□ den Wert

ABCDEF

enthaelt, so werden durch Teilstring-Angaben folgende String-Werte spezifiziert:

A□(1:4)	ABCD
A□(3:3)	C
A□(2:5)	BCDE

Eine Teilstring-Angabe loest keine Ausnahmen aus. Sind also die Indizes m und n (siehe oben) nicht im Bereich von 1 bis LEN(A□) (LEN(A□) ist die Laenge des String-Wertes, der A□ zugeordnet ist), so gelten folgende Festlegungen:

- m = MAX(m,1) und n = MIN(n,LEN(A□))
  Der Anfangsindex m kann nicht kleiner als 1 und der Endeindex n kann nicht groesser als die Anzahl der Zeichen des String-Wertes werden.

- Gilt nach dieser Korrektur von m und n die Beziehung m > n, so spezifiziert die Teilstring-Angabe einen leeren String-Wert. Gilt m <= LEN(A□), so handelt es sich um den leeren String-Wert, der unmittelbar vor dem m-ten Zeichen von A□ steht. Gilt m > LEN(A□), so handelt es sich um den leeren String-Wert, der unmittelbar an A□ angehaengt ist.

Demzufolge gilt (siehe oben)

A□(-1:10)	ABCDEF
A□(5:10)	EF
A□(3:2)	leerer String-Wert, der vor dem Zeichen C steht
A□(8:10)	leerer String-Wert, der auf das Zeichen F folgt

Soll eine indizierte String-Variable eine Teilstring-Angabe erhalten, so ist die Teilstring-Angabe nach den Indizes anzugeben. Die indizierte Variable MATRIX□(I,J) ist dann also als MATRIX(I,J)(3:5) und nicht etwa als MATRIX(3:5)(I,J) zu notieren. Bei Betreten einer Programmeinheit besitzen saemtliche in ihr definierten String-Variablen einen undefinierten Wert. Wird auf eine String-Variable zugegriffen, bevor ihr ein Wert zugewiesen wurde, so wird ihr ein leerer String-Wert zugeordnet und eine triviale Ausnahme ausgeloest.

6.3. String-Ausdruecke

String-Ausdruecke werden aus String-Operanden (Konstanten, Variable, Funktionsrufe) und dem Operator fuer Verkettung gebildet. Ihre Berechnung liefert jeweils einen einfachen String-Wert.

Syntax
------

ausdruck ::= string_ausdruck

string_ausdruck ::= string_primaeres [& string_primaeres]*

string_primaeres ::= <
- string_konstante
- string_variable
- string_funktionsruf
- (string_ausdruck)

>

string_funktionsruf ::=

< string_standardfunktion | string_nutzerfunktion > [({funktionsargument},)*]

Der Wert eines String-Ausdruckes wird durch Verkettung der Werte der String-Primaeren gebildet. Gilt z.B. A$ = "<" und B$ = "=", so gilt

A$ & B$ = "<=" und B$ & A$ = "=<".

Folgen mehrere Verkettungsoperatoren aufeinander, so erfolgt die Abarbeitung von links nach rechts, falls nicht unter Ausnutzung der Assoziativitaet des Verkettungsoperators eine andere Berechnungsreihenfolge festgelegt wird, um eine optimale Abarbeitung zu erzielen. Durch die Verwendung von Klammern kann die Abarbeitungsreihenfolge anders festgelegt werden. Ist ein Operand einer Verkettungsoperation ein geklammerter Ausdruck, so wird zunaechst der Wert des geklammerten Ausdrucks berechnet und anschliessend die Verkettung ausgefuehrt.
Tritt in einem String-Ausdruck eine String-Variable auf, so werden zunaechst die Indizes berechnet, falls es sich um eine indizierte Variable handelt. Anschliessend werden die Indizes der Teilstring-Angabe berechnet, falls eine solche vorhanden ist. Daran anschliessend wird mit diesen Informationen der String-Wert ermittelt, der in die Berechnung eingeht.
Der Aufruf einer String-Funktion bewirkt die Unterbrechung der Berechnung des Ausdrucks, die Bereitstellung der Funktionsargumente und anschliessend die Abarbeitung der Funktion (s. Abschn. 9.). Nach Abarbeitung der Funktion identifiziert die Funktionsbezeichnung einen einfachen String-Wert, mit dem die Berechnung des Ausdrucks fortgesetzt wird.
Die Anzahl und die Typen der Argumente in einem Funktionsaufruf muessen mit der Anzahl und den Typen der korrespondierenden formalen Parameter in der Funktionsdefinition uebereinstimmen. Ist als Funktionsargument ein Feld angegeben, so muss es die gleiche Zahl von Dimensionen wie der korrespondierende Parameter haben. Die Parametervermittlung erfolgt nach dem Prinzip "call by value" (s. Abschn. 9.).
Wird eine externe String-Funktion gerufen, so muss die Arithmetik-Option dieser Funktion mit der Arithmetik-Option der rufenden Programmeinheit uebereinstimmen, falls als Parameter numerische Werte oder Felder uebergeben werden.

Jede in einem Ausdruck gerufene Nutzerfunktion muss in der gleichen Programmeinheit zuvor als Nutzerfunktion ausgewiesen werden. Dies kann entweder durch die Definition der Funktion (interne Funktionsdefinition, s. Abschn. 9.) oder durch Angabe der Funktion in einer declare-Anweisung erfolgen. Die Angabe in einer declare-Anweisung ist notwendig, wenn die gerufene Funktion extern, d.h. eine nicht in der vorliegenden Programmeinheit definierte Funktion ist bzw. wenn eine interne Funktion vor ihrer Definition gerufen werden soll. Standardfunktionen sind a priori bekannt und koennen ohne besondere Vorkehrungen gerufen werden. Wird bei der Berechnung eines String-Ausdruckes ein String-Wert erzeugt, der mehr als 32767 Zeichen enthaelt, so wird eine nicht triviale Ausnahme ausgeloest.
Zu beachten ist weiterhin, dass das Zeichen & sowohl als Verkettungsoperator als auch als Zeichen fuer Fortsetzungszeilen fungiert. Durch die Anweisungen

```
100 PRINT "ABC" &&
& "XYZ"
110 PRINT "ABC" &
&& "XYZ"
```

wird deshalb der String-Wert ABCXYZ gedruckt, waehrend

```
120 PRINT "ABC" &
& "XYZ"
```

zu einem syntaktischen Fehler fuehrt.

6.4. String-Standardfunktionen

Eine Reihe haeufig benoetigter Algorithmen werden als Standardfunktionen bereitgestellt. Es werden in diesem Abschnitt auch einige numerische Standarfunktionen erlaeutert, deren Argumente String-Werte sind.

Syntax

```
string_standardfunktion ::=
        CHR¤ | DATE¤ | LCASE¤ | LTRIM¤ | REPEAT¤ | RTRIM¤ |
        STR¤ | TIME¤ | UCASE¤ | USING¤

num_standardfunktion ::=
        LEN | ORD | POS | VAL

                                 ┌                                 ┐
                                 │ einfache_string_variable        │
num_funktionsruf ::= MAXLEN (<   │                                 │>)
                                 │ string_feld                     │
                                 └                                 ┘
```

In der folgenden Tabelle werden die oben aufgefuehrten Standardfunktionen naeher erlaeutert. Dabei steht M fuer einen Index, d.h. einen ggf. gerundeten ganzzahligen numerischen Wert, X fuer einen numerischen Ausdruck. V¤ repraesentiert eine einfache String-Variable oder ein String-Feld und A¤ und B¤ stehen fuer String-Ausdruecke.

Tabelle 2: String-Standardfunktionen

Funktion	Funktionswert
CHR□(M)	String-Wert, bestehend aus einem Zeichen; dieses Zeichen nimmt die M-te Position im verwendeten Zeichensatz ein; M muss zwischen Null und 127 (Anzahl der Zeichen des Zeichensatzes) liegen (s. Anl. 2); CHR□(53) = "5" , CHR□(65) = "A"
DATE□	Datum in der Form "JJJJMMTT", wobei JJJJ fuer das Jahr, MM fuer den Monat und TT fuer den Tag steht; fuer den 9. Mai 1987 gilt DATE□ = "19870509"
LCASE□(A□)	String-Wert, der aus dem Wert A□ dadurch erzeugt wird, dass alle darin enthaltenen Grossbuchstaben durch Kleinbuchstaben ersetzt werden; LCASE□("Name") = "name"
LEN(A□)	Anzahl der Zeichen des String-Wertes von A□; LEN("X") = 1, LEN(""") = 1, LEN("") = 0
LTRIM□(A□)	String-Wert, der aus dem Wert von A□ dadurch erzeugt wird, dass fuehrende Leerzeichen gestrichen werden; LTRIM□(" ABC") = "ABC", LTRIM□("XYZ") = "XYZ"
MAXLEN(V□)	maximale Zeichenzahl fuer String-Werte, die der einfachen String-Variablen V□ bzw. einem Element des String-Feldes V□ zugewiesen werden koennen
ORD(A□)	Position des durch A□ beschriebenen Zeichens im verwendeten Zeichensatz, wobei das erste Zeichen des Zeichensatzes die Position Null hat; moegliche Werte von A□ sind einzelne Zeichen des Zeichensatzes oder String-Werte aus zwei oder drei Zeichen, die eine mnemonische Darstellung eines Zeichens des Zeichensatzes bilden (s. Anl. 2); bei Mnemoniks koennen Gross- und Kleinbuchstaben verwendet werden; ORD("A") = 65, ORD("a") = 97, ORD("5") = 53, ORD("CR") = 13, ORD("LCA") = 97 ORD("ABC") loest eine nicht triviale Ausnahme aus
POS(A□,B□)	Anfangsposition des ersten Auftretens des String-Wertes von B□ (Teilstring) innerhalb des String-Wertes von A□; tritt B□ in A□ nicht auf, so ist das Ergebnis Null; POS(A□,"") liefert stets 1; POS("ABCD","BC") = 2, POS("XYZ","A") = 0

Tabelle 2: Fortsetzung

1	2
POS(A□,B□,M)	Anfangsposition des ersten Auftretens des String-Wertes von B□ (Teilstring) innerhalb des String-Wertes von A□, wobei die Suche in A□ ab dem M-ten Zeichen beginnt; die Funktion liefert Null, falls B□ nicht in dem entsprechenden Teil von A□ auftritt oder M groesser als LEN(A□) ist; falls A□ = "ABCABCD" gilt, so gilt POS(A□,"B",1) = 2, POS(A□,"B",3) = 5, POS(A□,"B",6) = 0 der Funktionsruf POS(A□,"",M) liefert immer MAX(M,1), falls M <= LEN(A□) gilt
REPEAT□(A□,M)	String-Wert, der aus M Verkettungen des String-Wertes von A□ entsteht; M muss groesser oder gleich Null sein
RTRIM□(A□)	String-Wert, der aus dem Wert von A□ dadurch erzeugt wird, dass abschliessende Leerzeichen gestrichen werden; RTRIM□("ABC ") = "ABC" , RTRIM□("XYZ") = "XYZ"
STR□(X)	Repraesentation des numerischen Wertes X in der gleichen Form wie sie bei einer print-Anweisung erzeugt wird, wobei allerdings fuehrende und abschliessende Leerzeichen nicht auftreten; STR□(123.5) = "123.5" , STR□(-3.14) = "-3.14" , STR□(1.0E+1) = "10"
TIME□	Zeit in der Form "HH:MM:SS", wobei HH die Stunde, MM die Minute und SS die Sekunde darstellt; um 11.15 Uhr gilt TIME□ = "11:15:00"
UCASE□(A□)	String-Wert, der aus dem Wert von A□ dadurch erzeugt wird, dass alle darin enthaltenen Kleinbuchstaben durch Grossbuchstaben ersetzt werden; UCASE□("Name") = "NAME"
USING□(A□,X)	Repraesentation des numerischen Wertes X in der gleichen Form wie sie bei einer print-Anweisung mit using-Angabe (s. Abschn. 10.4.) unter Verwendung von A□ als Formatbeschreibung erzeugt wird; die im Abschnitt 10.4. beschriebenen Ausnahmen bei formatierter Ausgabe koennen auch bei Anwendung der USING□-Funktion ausgeloest werden
VAL(A□)	Wert der numerischen Konstanten, die durch den String-Wert von A□ repraesentiert wird, wobei fuehrende und nachfolgende Leerzeichen in A□ ignoriert werden; VAL(" 123.5 ") = 123.5, VAL("0.0005e2") = 0.05 VAL("IX") loest eine nicht triviale Ausnahme aus

In den folgenden Faellen werden beim Aufruf der Standardfunktionen nicht triviale Ausnahmen ausgeloest:

- Das Argument der Funktion VAL repraesentiert keine numerische Konstante.

- Das Argument der Funktion VAL repraesentiert eine numerische Konstante die groesser als MAXNUM ist.

- Das Argument der Funktion CHR□ liegt nicht im Bereich von Null bis 127.

- Das Argument der Funktion ORD ist weder ein zulaessiges einzelnes Zeichen noch ein zulaessiges Mnemonik.

- Der Wert des zweiten Argumentes von REPEAT□ ist kleiner als Null.

6.5. Let-Anweisung fuer String-Werte

Die let-Anweisung fuer String-Werte gestattet die Zuweisung eines durch einen String-Ausdruck berechneten String-Wertes an eine oder mehrere String-Variable.

Syntax

```
let_anweisung ::= string_let_anweisung

string_let_anweisung ::=
                                *
      LET {string_variable}, = string_ausdruck
```

Die Werte der Variablen links vom Gleichheitszeichen (Zielvariable) werden durch den Wert des Ausdrucks rechts vom Gleichheitszeichen ersetzt bzw. modifiziert. Diesen so geaenderten Wert behalten die Zielvariablen solange, bis ihnen ein neuer Wert zugewiesen wird.
Das Schluesselwort LET muss im Gegensatz zu anderen BASIC-Versionen stets angegeben werden.
Die Abarbeitung der string-let-Anweisung erfolgt in folgenden Schritten:

- Die Indizes der indizierten Zielvariablen und die Indizes der Teilstring-Angaben der Zielvariablen werden in der Reihenfolge von links nach rechts berechnet.

- Der Wert des String-Ausdruckes rechts vom Gleichheitszeichen wird berechnet.

- Der berechnete Wert wird zur Aenderung des Wertes der Zielvariablen von links nach rechts herangezogen, wobei zwei Faelle unterschieden werden. Besitzt die Zielvariable keine Teilstring-Angabe, so ersetzt der berechnete Wert den bisherigen Wert der Zielvariablen vollstaendig. Besitzt die Zielvariable eine Teilstring-Angabe, so ersetzt der berechnete Wert den durch die Teilstring-Angabe spezifizierten Teil des bisherigen Wertes der Zielvariablen.

Durch die Anweisung

```
LET A¤,B¤ = "ABCD"
```

wird den Variablen A¤ und B¤ der String-Wert ABCD zugewiesen. Fuer diesen Wert von A¤ fuehren die folgenden Anweisungen zu angegebenen Resultaten

```
LET A¤(2:3) = "XY"          A¤ = "AXYD"
LET A¤(2:3) = ""            A¤ = "AD"
LET A¤(2:3) = A¤(1:2)       A¤ = "AABD"
LET A¤(2:1) = "X"           A¤ = "AXBCD"
```

Die Reihenfolge der Zielvariablen kann u. U. von Bedeutung sein. Gilt A¤ = "ABCD" so fuehren die folgenden Anweisungen zu verschiedenen Resultaten.

```
LET A¤(1:2),A¤(2:3) = "X"      A¤ = "XX"
LET A¤(2:3),A¤(1:2) = "X"      A¤ = "XD"
```

Wird bei der Aenderung des berechneten Wertes einer Zielvariablen festgestellt, dass der geaenderte Wert die maximal zulaessige Zeichenzahl fuer diese Zielvariable ueberschreitet, so wird eine nicht triviale Ausnahme ausgeloest.

6.6. Deklaration von String-Variablen

Durch eine declare-Anweisung koennen String-Variable explizit deklariert werden. Dabei kann die maximale Laenge der String-Werte festgelegt werden, die in den Variablen gespeichert werden koennen.

Syntax

```
declare_anweisung ::= DECLARE typ_deklaration

typ_deklaration ::=    _                                                  _ *
                      |                                                  |
STRING [*ganze_zahl] <  einfache_string_variable[*ganze_zahl]             >
                      |_                                                 _| ,
```

Mit einer declare-Anweisung koennen einfache String-Variable deklariert werden.
Die maximale Laenge der String-Werte, die einer in einer declare-Anweisung deklarierten String-Variablen zugewiesen werden koennen, wird wie folgt bestimmt:

1. Ist unmittelbar an die Variable durch * eine ganze Zahl angehaengt, so legt diese Zahl die Maximallaenge fest.

2. Trifft Fall 1 nicht zu, so wird die Laenge verwendet, die durch * an das Schluesselwort STRING angehaengt ist.

3. Treffen weder Fall 1 noch Fall 2 zu, so wird die standardmaessige Laenge von 32767 Zeichen verwendet.
   Eine Laengenangabe muss zwischen Null und 32767 liegen.

Durch die Anweisung

```
DECLARE STRING*50 A¤,B¤*30, C¤,D¤*25
```

wird festgelegt, dass den Variablen A¤ und C¤ String-Werte von maximal 50 Zeichen, der Variablen B¤ String-Werte von maximal 30 Zeichen und der Variablen D¤ String-Werte von maximal 25 Zeichen zugewiesen werden koennen.
Durch die Anweisung

```
DECLARE STRING X¤,Y¤*10,Z¤
```

wird festgelegt, dass den Variablen X¤ und Z¤ String-Werte von maximal 32767 Zeichen und der Variablen Y¤ String-Werte von maximal 10 Zeichen zugewiesen werden koennen.
Die explizite Deklaration einfacher String-Variablen in einer declare-Anweisung ist zwingend nur dann notwendig, wenn die Maximallaenge fuer die zu speichernden String-Werte kleiner als 32767 sein soll. Ist dies nicht der Fall, so koennen einfache String-Variable auch implizit durch ihr Auftreten im Programm deklariert werden.
Eine Laengenangabe von 0 in einer Deklaration fuer eine String-Variable bewirkt, dass dieser String-Variablen nur ein String-Wert der Laenge 0, d.h. ein leerer String-Wert zugeordnet werden kann.

7. Felder

Felder sind geordnete Folgen von Werten der gleichen Wertart (numerisch oder string). Jedes Element eines Feldes repraesentiert einen einzelnen Wert. Die Auswahl eines Elementes erfolgt durch Angabe von Indizes. Feldelemente werden deshalb auch als indizierte Variable bezeichnet.
In einem BASIC-Programm koennen ein-, zwei- und dreidimensionale Felder deklariert werden. Ueber die in den Abschnitten 5 und 6 beschriebenen Ausdrucksmittel koennen Feldelemente, d.h. indizierte Variable verarbeitet werden.

7.1. Deklaration von Feldern

Jedes Feld muss vor seiner Verwendung explizit deklariert werden. Durch die Deklaration werden die Anzahl der Dimensionen und die Indexgrenzen jeder Dimension festgelegt. Die declare-Anweisung bietet gegenueber der dim-Anweisung erweiterte Moeglichkeiten zur Deklaration von Feldern.

Syntax

```
                                  ⌈                          ⌉*
                                  |  num_feld_deklaration     |
dim_anweisung ::= DIM <           |                          |>
                                  |  string_feld_deklaration  |
                                  ⌊                          ⌋,

                                                            3
num_feld_deklaration ::= num_feld ({indexgrenzen},)

                                                            3
string_feld_deklaration ::= string_feld ({indexgrenzen},)

                                  ⌈                                ⌉
                                  |  untere_grenze TO obere_grenze  |
indexgrenzen ::= <                |                                |>
                                  |  obere grenze                   |
                                  ⌊                                ⌋

                                  ⌈ ⌉
                                  |  +  |
untere_grenze ::=                 |  -  |  ganze_zahl
                                  ⌊ ⌋

                                  ⌈ ⌉
                                  |  +  |
obere_grenze ::=                  |  -  |  ganze_zahl
                                  ⌊ ⌋

                                                  *
option_anweisung ::= OPTION {option},
```

```
                    ┌ ┐
                    │ 0 │
option ::= BASE <   │   │>
                    │ 1 │
                    └ ┘

declare_anweisung ::= DECLARE typ_deklaration

                                                    *
typ_deklaration ::= NUMERIC {num_feld_deklaration},

typ_deklaration ::=                                          *
    STRING[*ganze_zahl]{string_feld_deklaration[*ganze_zahl]},

                                 ┌                          ┐
                                 │ MAXSIZE (feld)           │
                                 │ SIZE (feld [,index])     │
num_funktionsaufruf ::= <        │                          │>
                                 │ LBOUND (feld [,index])   │
                                 │ UBOUND (feld [,index])   │
                                 └                          ┘
```

Jedes Feld, dass nicht formaler Parameter einer Funktion bzw. Prozedur ist, muss vor seiner ersten Verwendung mit Hilfe einer dim- bzw. declare-Anweisung explizit deklariert werden. Bei Feldern, die formale Parameter von Funktionen bzw. Prozeduren sind, gilt ihr Auftreten in der Parameterliste als Deklaration. Jedes Feld darf innerhalb einer Programmeinheit nur einmal deklariert werden. Die implizite Deklaration von Feldern ist nicht moeglich. Durch eine Felddeklaration werden fuer ein Feld die Anzahl der Dimensionen (eine, zwei, oder drei) und die Indexgrenzen fuer jede Dimension festgelegt. Alle Zugriffe auf ein Feld bzw. seine Elemente muessen in der Dimensionsanzahl mit der zugehoerigen Felddeklaration uebereinstimmen.
Durch die Indexgrenzen wird festgelegt, in welchem Bereich die Indizes in der entsprechenden Dimension liegen duerfen. Die untere und die obere Grenze sind in diesem Bereich eingeschlossen. Wird bei der Festlegung der Indexgrenzen die untere Grenze nicht explizit angegeben, so ist sie in Abhaengigkeit von der sog. BASE-Option entweder 1 oder 0. Die obere Grenze muss stets groesser oder gleich der unteren Grenze sein.
Durch die Anweisung

```
100 DIM X(-5 TO 5) , Y(10)
```

werden zwei eindimensionale numerische Felder (Vektoren) definiert. Das Feld X besitzt 11 Elemente, die mit den Indizes von -5 bis +5 indiziert werden. Das Feld Y besitzt in Abhaengigkeit von der BASE-Option entweder 10 oder 11 Elemente. Besitzt die BASE-Option den Wert 1, so besitzt das Feld 10 Elemente mit den Indizes von 1 bis 10. Besitzt die BASE-Option den Wert 0, so besitzt das Feld 11 Elemente mit den Indizes von 0 bis 10.
Durch die Anweisung

```
110 DIM TEXT¤(1 TO 10 , 1 TO 10)
```

wird ein zweidimensionales string-Feld (Matrix) mit dem Namen TEXT¤ deklariert, das 100 Elemente umfasst. Die Indizes in beiden

Dimensionen koennen die Werte 1 bis 10 annehmen.

Die BASE-Option kann nur die Werte 0 und 1 annehmen. Die Festlegung dieses Wertes erfolgt in einer option-Anweisung , z.B. durch

```
10 OPTION BASE 0
```

Erfolgt keine solche Festlegung, so hat die BASE-Option implizit den Wert 1. Die BASE-Option ist lokal in jeder Programmeinheit gueltig und darf in jeder Programmeinheit hoechstens einmal durch eine option-Anweisung eingestellt werden. Diese option-Anweisung muss vor der ersten dim- bzw. declare-Anweisung stehen.
Die Anwendung der BASE-Option als implizite untere Grenze bei der Deklaration von Feldern sollte moeglichst vermieden werden. Die explizite Angabe der unteren Grenze verbessert die Lesbarkeit des Programmtextes. Die Moeglichkeit der Verwendung der BASE-Option wurde aus Kompatibilitaetsgruenden zu anderen BASIC-Sprachversionen vorgesehen.
Die oben als Beispiel mit Hilfe von dim-Anweisungen deklarierten Felder koennen wie folgt mit declare-Anweisungen deklariert werden :

```
100 DECLARE NUMERIC X(-5 TO 5) , Y(10)
110 DECLARE STRING TEXT$(1 TO 10 , 1 TO 10)
```

Die declare-Anweisung bietet damit alle Ausdrucksmoeglichkeiten, die auch die dim-Anweisung bietet. Darueberhinaus ermoeglicht sie, bei der Deklaration von string-Feldern die Maximallaenge der string-Werte, die in den Feldelementen gespeichert werden koennen, explizit festzulegen.

```
120 DECLARE STRING*50 PREIS$(100 TO 150) * 20 , &
&                     BEZEICHNUNG$(100 TO 150)
```

Die Bestimmung der Maximallaenge erfolgt nach dem gleichen Prinzip wie im Abschnitt 6.6. beschrieben. Daraus folgt, dass die Elemente des Feldes PREIS$ string-Werte von maximal 20 Zeichen und die Elemente des Feldes BEZEICHNUNG$ string-Werte von maximal 50 Zeichen aufnehmen koennen. Die Elemente des oben deklarierten Feldes TEXT$ koennen Werte bis zu 32767 Zeichen aufnehmen. Die Laengenangaben in der declare-Anweisung muessen zwischen Null und 32767 liegen.
Wird bei der Programmabarbeitung eine Zeile erreicht, die eine dim-, declare- oder option-Anweisung enthaelt, so wird diese Zeile ohne Wirkung uebergangen.
Bei der Deklaration eines Feldes wird fuer jede Dimension durch Angabe der Indexgrenzen ein Umfang der Dimension festgelegt, der sich aus

```
Umfang = obere Grenze - untere Grenze + 1
```

ergibt. Die bei der Deklaration festgelegte Anzahl der Elemente eines Feldes ist das Produkt der Umfaenge der einzelnen Dimensionen. Eine Reihe von Anweisungen koennen eine sog. Redimensionierung eines Feldes vornehmen. Darunter wird die Moeglichkeit verstanden, zur Programmlaufzeit den Umfang einer oder mehrerer Dimensionen bzw. die unteren und/oder oberen Grenzen zu veraendern. Die Anzahl der Dimensionen kann durch eine Redimensionierung allerdings nicht veraendert werden. Die folgenden Funktionen dienen dazu, die aktuellen Werte fuer den Umfang eines Feldes, bzw. einer Dimension und fuer die unteren oder oberen Grenzen

zur Laufzeit festzustellen. Dabei steht F fuer eine Feldbezeichnung (numerisches Feld oder string-Feld) und N fuer einen Index, d.h. fuer einen ggf. gerundeten ganzzahligen numerischen Wert. Die Beispiele beziehen sich auf die oben deklarierten Felder und setzen voraus, dass keine Redimensionierung erfolgte.

Funktion	Funktionswert
MAXSIZE(F)	maximale Zahl der Elemente eines Feldes Diese Zahl darf bei Redimensionierungen nicht ueberschritten werden. MAXSIZE(X) = 11, MAXSIZE(TEXT□) = 100
SIZE(F)	aktuelle Zahl der Elemente eines Feldes Es gilt immer SIZE(F) <= MAXSIZE(F). Bei Betreten einer Programmeinheit gilt fuer alle in ihr deklarierten Felder SIZE(F) = MAXSIZE(F) SIZE(PREIS□) = 51
SIZE(F,N)	aktueller Umfang der N-ten Dimension (N = 1,2,3) SIZE(X,1) = 11, SIZE(TEXT□,1) = 10 SIZE(TEXT□,2) = 10
LBOUND(F,N)	aktuelle untere Grenze fuer die N-te Dimension (N = 1,2,3) LBOUND(X,1) = -5 , LBOUND(TEXT□,2) = 1
LBOUND(F)	aktuelle untere Grenze; F muss eindimensional sein LBOUND(X) = -5 , LBOUND(PREIS□) = 100
UBOUND(F,N)	aktuelle obere Grenze fuer die N-te Dimension (N = 1,2,3) UBOUND(X,1) = 5 , UBOUND(TEXT□,2) = 10
UBOUND(F)	aktuelle obere Grenze, F muss eindimensional sein UBOUND(X) = 5 , UBOUND(PREIS□) = 150

Bei den Funktionsaufrufen von SIZE, LBOUND und UBOUND, bei denen ein Index als zweiter Parameter verwendet wird, wird vorausgesetzt, dass dieser Index nicht kleiner als 1 und nicht groesser als die Zahl der Dimensionen des auftretenden Feldes ist. Anderenfalls wird eine nicht triviale Ausnahme ausgeloest. Die Dimensionen eines Feldes werden von links nach rechts, beginnend mit 1 gezaehlt.

8. Steueranweisungen und Steuerstrukturen

Die Anweisungen eines BASIC-Programms werden normalerweise sequentiell, d.h. Zeile fuer Zeile abgearbeitet. Durch Steuerstrukturen bzw. Steueranweisungen kann diese sequentielle Reihenfolge unterbrochen werden.

8.1. Logische Ausdruecke

In einigen Steueranweisungen bzw. -strukturen erfolgt anhand von Bedingungen die Entscheidung, wie die weitere Programmabarbeitung verlaeuft. Zum Beispiel kann durch eine Bedingung angegeben werden, wie oft eine Schleife abgearbeitet werden soll. Bedingungen werden durch logische Ausdruecke formuliert.

Syntax

```
                                                                  *
logischer_ausdruck ::= logischer_term [OR logischer_term]
                                                                  *
logischer_term ::= logischer_faktor [AND logischer_faktor]

                                  ┌                        ┐
                                  │ vergleich              │
logischer_faktor ::= [NOT] <      │                        │>
                                  │ (logischer_ausdruck)   │
                                  └                        ┘

                  ┌                                                ┐
                  │ num_ausdruck  relation  num_ausdruck           │
vergleich ::= <   │                                                │>
                  │ string_ausdruck  relation  string_ausdruck     │
                  └                                                ┘

relation ::= = | <> | >< | < | > | <= | =< | >= | =>
```

Den Grundbestandteil logischer Ausdruecke bilden die Vergleiche von numerischen oder string-Ausdruecken. Es stehen sechs Vergleichsrelationen zur Verfuegung:

Relation	Darstellung durch
gleich	=
ungleich	<> oder ><
kleiner	<
kleiner gleich	<= oder =<
groesser	>
groesser gleich	>= oder =>

Bei einem Vergleich werden zunaechst die beiden Ausdruecke (linker und rechter Operand des Relationsoperators) berechnet und anschliessend die ermittelten Werte verglichen.
Der Vergleich numerischer Werte erfolgt anhand der Groesse dieser Werte.

Der Vergleich von string-Werten erfolgt Zeichen fuer Zeichen von links beginnend, bis einer der folgenden Faelle auftritt:

a) Die zwei untersuchten Zeichen z.B. C1¤ und C2¤ sind verschieden. Es wird das Zeichen als kleiner betrachtet, dessen Position im Zeichensatz kleiner ist, d.h es gilt:
C1¤ < C2¤ , falls ORD(C1¤) < ORD(C2¤)
Der string-Wert, der das kleinere Zeichen enthaelt, wird dann als kleiner betrachtet. Die restlichen Zeichen werden nicht untersucht.

b) Die string-Werte sind gleich lang und es wird das Ende erreicht, ohne dass differierende Zeichen gefunden wurden. Dann und nur dann sind beide Zeichenketten gleich.

c) Die string-Werte sind verschieden lang und es wird das Ende des kuerzeren string-Wertes erreicht, ohne dass differierende Zeichen gefunden werden. Dann wird die kuerzere Zeichenkette als kleiner betrachtet.

Die Relationen zwischen zwei string-Werten seien durch folgende Beispiele illustriert:

- "BASIC" = "BASIC" , da beide string-Werte gleich lang sind und identische Zeichenketten enthalten

- "Basic" > "BASIC" , da ORD("a") > ORD("A")

- "BASIC" < "BASIC " , da LEN("BASIC") < LEN("BASIC ")

Jeder Vergleich liefert als Ergebnis einen der Wahrheitswerte (logische Werte) "wahr" oder "falsch", je nachdem ob die durch den Relationsoperator vorgegebene Beziehung gilt oder nicht. Sei z.B. A=1.5 und B=20.187 sowie A¤="Basic" und B¤="BASIC", so liefern Vergleiche dieser Groessen folgende Wahrheitswerte.

Vergleich	Ergebnis	Vergleich	Ergebnis
A = B	falsch	A¤ = B¤	falsch
A <> B	wahr	A¤ <> B¤	wahr
A < B	wahr	A¤ < B¤	falsch
A <= B	wahr	A¤ <= B¤	falsch
A > B	falsch	A¤ > B¤	wahr
A >= B	falsch	A¤ >= B¤	wahr

Die logischen Operatoren NOT, AND und OR haben die uebliche Bedeutung. NOT ist ein monadischer Operator der auf einen logischen Wert V wie folgt wirkt:

V	NOT V
wahr	falsch
falsch	wahr

Die dyadischen Operatoren AND und OR haben folgende Wirkung:

V1	V2	V1 AND V2	V1 OR V2
wahr	wahr	wahr	wahr
wahr	falsch	falsch	wahr
falsch	wahr	falsch	wahr
falsch	falsch	falsch	falsch

Die Abarbeitungsreihenfolge der logischen Operatoren wird durch die Prioritaet festgelegt.

NOT	hoechste Prioritaet
AND	
OR	niedrigste Prioritaet

Dabei ist zu beachten, dass vor Ausfuehrung einer logischen Operation die Operanden berechnet werden muessen. Ist ein Operand ein Vergleich, so muss also zunaechst der Vergleich ausgefuehrt werden. Operatoren hoeherer Prioritaet werden vor Operatoren mit niederer Prioritaet abgearbeitet. Fuer die Bestimmung der Wahrheitswerte fuer logische Terme und logische Ausdruecke (mehrere aufeinander folgende Operatoren gleicher Prioritaet) gelten folgende Regeln:

- logischer Term
  Die logischen Faktoren werden von links nach rechts abgearbeitet, bis ein Faktor den Wert "falsch" liefert oder alle Faktoren mit "wahr" abgearbeitet werden. Im ersten Fall erhaelt der logische Term den Wert "falsch" und die restlichen Faktoren des Terms werden nicht mehr abgearbeitet. Im zweiten Fall erhaelt der logische Term den Wert "wahr".

- logischer Ausdruck
  Die logischen Terme werden von links nach rechts abgearbeitet, bis ein Term den Wert "wahr" liefert oder alle Terme mit "falsch" abgearbeitet werden. Im ersten Fall erhaelt der logische Ausdruck den Wert "wahr" und die restlichen Terme des Ausdrucks werden nicht mehr abgearbeitet. Im zweiten Fall erhaelt der logische Ausdruck den Wert "falsch".

Bei logischen Ausdruecken kann also im Gegensatz zu numerischen und string-Ausdruecken der Fall eintreten, dass Teile des Ausdrucks nicht abgearbeitet werden. Im Falle von logischen Ausdruecken wird deshalb auch keine Umordnung der Operatoren vorgenommen bzw. keine andere Berechnungsreihenfolge festgelegt. Fuer geklammerte Ausdruecke gelten die ueblichen Festlegungen, d.h. ist ein Operand einer logischen Operation ein geklammerter Ausdruck, so wird zunaechst der Wert dieses Ausdrucks berechnet und anschliessend die Operation ausgefuehrt.
In den folgenden Beispielen wird durch vollstaendige Klammerung die Abarbeitungsreihenfolge verdeutlicht.

Ausdruck	Abarbeitungsreihenfolge
X <= Y AND Y <= Z	(X <= Y) AND (Y <= Z)
NOT X <= Y AND Y <= Z	(NOT(X <= Y)) AND (Y <= Z)
NOT(X <= Y AND Y <= Z)	NOT((X <= Y) AND (Y <= Z))
X <= Y AND Y <= Z OR Y > 0	((X <=Y) AND (Y <= Z)) OR (Y > 0)
X <= Y AND (Y <= Z OR Y > 0)	(X <= Y) AND ((Y <=Z) OR (Y > 0))

Die Tatsache, dass bei logischen Ausdruecken Teilausdruecke u.U. nicht abgearbeitet werden, muss beim Programmieren beachtet werden. So wird z.B. bei

```
A < B AND STATUS (X) = 0
```

die Funktion STATUS nur aufgerufen, wenn A < B gilt. In dieser Funktion duerfen also keine Aktionen vorhanden sein, die den weiteren Programmablauf beeinflussen.
Andererseits kann diese Tatsache auch bewusst genutzt werden. Ist A ein Vektor, dann wird durch den Ausdruck

```
LBOUND(A) <= I AND I <= UBOUND(A) AND A(I) = X
```

gesichert, dass der Vergleich A(I) = X nur ausgefuehrt wird, wenn der Index I innerhalb des zulaessigen Bereiches liegt. Eine diesbezuegliche Ausnahme kann also nicht auftreten.

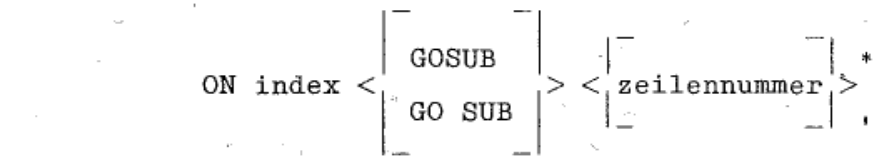
8.2. Steueranweisungen

Die Steueranweisungen ermoeglichen die Unterbrechung der normalen sequentiellen Abarbeitungsreihenfolge durch Vorgabe von Zeilennummern, die die naechste abzuarbeitende Zeile markieren. Alle diese Zeilennummern muessen sich in der gleichen Programmeinheit wie die Steueranweisung befinden.

Syntax

```
goto_anweisung ::= <[ GOTO  ]> zeilennummer
                   <[ GO TO ]>

on_goto_anweisung ::=
    ON index <[ GOTO  ]> <[ zeilennummer ]>*
             <[ GO TO ]>
        [ELSE imperative_anweisung]

gosub_anweisung ::= <[ GOSUB  ]> zeilennummer
                    <[ GO SUB ]>
```

on_gosub_anweisung ::=



[ELSE imperative_anweisung]

return_anweisung ::= RETURN

Die Ausfuehrung einer goto_anweisung bewirkt die Uebergabe der Steuerung an die Zeile, deren Zeilennummer auf das Schluesselwort GOTO folgt. Durch die Anweisung

```
GOTO 320
```

erfolgt also ein Sprung zu der Anweisung auf der Zeile 320.
Die on-goto-Anweisung bewirkt die Uebergabe der Steuerung an eine ausgewaehlte Zeile. Zunaechst wird der Index berechnet und sein Wert zur Auswahl einer Zeilennummer aus der Liste der Zeilennummern, die auf das Schluesselwort GOTO folgen, benutzt. Die Zeilennummern in der Liste werden von links nach rechts mit 1 beginnend durchnumeriert. Die letzte Zeilennummer in der Liste sei die m-te Zeilennummer. Besitzt der Index einen Wert zwischen 1 und m, so wird die entsprechende Zeilennummer ausgewaehlt und die Programmabarbeitung auf der entsprechenden Zeile fortgesetzt.
Ist der Index kleiner als 1 oder groesser als m, so wird die nach ELSE stehende Anweisung ausgefuehrt bzw. eine nicht triviale Ausnahme ausgeloest, falls kein ELSE-Teil vorhanden ist. Uebergibt die nach ELSE stehende Anweisung die Steuerung nicht an eine andere Zeile, so wird nach ihrer Abarbeitung die naechste, auf die on-goto-Anweisung folgende Zeile abgearbeitet.
Die Anweisung

```
1000 ON I GOTO 510,130,480,1200 ELSE LET I = 0
1010 ...
```

bewirkt, dass ein Sprung zur Zeile 510, 130, 480 bzw. 1200 erfolgt, falls der gerundete Wert der Variable I 1, 2, 3 oder 4 ist. Anderenfalls wird die nach ELSE stehende let-Anweisung und anschliessend die Anweisung auf der Zeile 1010 abgearbeitet.
Mit Hilfe von gosub-, on-gosub- und return-Anweisungen koennen einfache Unterprogramme organisiert werden. Diese Anweisungen sind fuer viele BASIC-Sprachversionen typisch. Das zugrunde liegende Konzept ist jedoch veraltet. Unterprogramme koennen weitaus besser mit den im Abschnitt 9. beschriebenen Ausdrucksmitteln organisiert werden.
Die Ausfuehrung von gosub-, on-gosub- und return-Anweisungen kann mit Hilfe eines speziellen Speichers (Kellerspeicher) fuer Zeilennummern beschrieben werden. Jede Programmeinheit und jede interne Prozedur (s. Abschn. 9.) besitzt einen solchen Kellerspeicher. Bei Betreten einer Programmeinheit oder einer internen Prozedur ist der zugeordnete Speicher stets leer.
Die Ausfuehrung einer gosub-Anweisung bewirkt zunaechst die Abspeicherung der Zeilennummer der Zeile, in der sich die gosub-Anweisung befindet, in den entsprechenden Zeilennummernspeicher. Anschliessend erfolgt wie bei der goto-Anweisung die Uebergabe der Steuerung an die Zeile, deren Zeilennummer auf das Schluesselwort GOSUB folgt.

Durch die Anweisung

```
150 GOSUB 400
```

wird also die Zeilennummer 150 in den Zeilennummernspeicher eingetragen. Anschliessend erfolgt ein Sprung zur Zeile 400.

Die Ausfuehrung einer on-gosub-Anweisung bewirkt die Uebergabe der Steuerung an eine ausgewaehlte Zeile. Zunaechst wird der Index berechnet und sein Wert zur Auswahl einer Zeilennummer aus der Liste der Zeilennummern, die auf das Schluesselwort GOSUB folgen, benutzt. Die Zeilennummern in der Liste werden von links nach rechts mit 1 beginnend durchnumeriert. Die letzte Zeilennummer in der Liste sei die m-te Zeilennummer. Besitzt der Index einen Wert zwischen 1 und m, so wird zunaechst die Zeilennummer der Zeile, in der sich die on-gosub-Anweisung befindet, in den Zeilennummernspeicher abgespeichert. Anschliessend wird die entsprechende Zeilennummer aus der Liste ausgewaehlt und die Programmabarbeitung auf dieser Zeile fortgesetzt.
Ist der Index kleiner als 1 oder groesser als m, so wird die nach ELSE stehende Anweisung ausgefuehrt bzw. eine nicht triviale Ausnahme ausgeloest, falls kein ELSE-Teil vorhanden ist. Uebergibt die nach ELSE stehende Anweisung die Steuerung nicht an eine andere Zeile, so wird nach ihrer Abarbeitung die naechste, auf die on-gosub-Anweisung folgende Zeile abgearbeitet.
Die Anweisung

```
500 ON I GOSUB 600,700,800 ELSE PRINT I
510 ...
```

fuehrt in Abhaengigkeit vom Wert der Variablen I zu folgenden Aktionen. Besitzt I den Wert 1, 2 oder 3, so wird die Zeilennummer 500 in den Zeilennummernspeicher eingetragen und anschliessend ein Sprung zur Zeile 600, 700 bzw. 800 ausgefuehrt. Anderenfalls wird der Wert von I ausgegeben und anschliessend die Anweisung auf der Zeile 510 abgearbeitet.

Eine return-Anweisung bewirkt, dass aus dem entsprechenden Zeilennummernspeicher die Zeilennummer, die zuletzt eingetragen wurde, entnommen wird. Anschliessend wird die Abarbeitung mit der Zeile fortgesetzt, die auf die Zeile mit der entnommenen Zeilennummer folgt. Die Entnahme der Zeilennummer aus dem Zeilennummernspeicher bedeutet, dass sie anschliessend in dem Speicher nicht mehr vorhanden ist. Ist bei Ausfuehrung einer return-Anweisung der Zeilennummernspeicher leer, so wird eine nicht triviale Ausnahme ausgeloest.

Innerhalb einer internen Prozedur arbeiten gosub-, on-gosub- und return-Anweisungen stets mit dem Zeilennummernspeicher, der dieser internen Prozedur zugeordnet ist. Ansonsten verwenden sie den Zeilennummernspeicher der betreffenden Programmeinheit.

Bei Verlassen einer Programmeinheit bzw. einer internen Prozedur geht der zugeordnete Zeilennummernspeicher verloren. Es ist nicht notwendig, dass der Speicher zu diesem Zeitpunkt leer ist.

Es sei angenommen, dass in einem Programm an mehreren Stellen die Variablen A, B, C und D initialisiert werden muessen. Dies kann unter Verwendung von gosub-Anweisungen und einer return-Anweisung erfolgen.

```
  .
  .
  .
100 PRINT X
110 GOSUB 500
120 LET X = 0.5
  .
  .
  .
230 PRINT X * Y
240 GOSUB 500
250 LET X,Y = 1
  .
  .
  .
500 REM Initialisierung der Variablen A, B, C und D
510 LET A,B = 1
520 LET C = 0.5
530 LET D = 0
540 RETURN
  .
  .
  .
```

Nach Abarbeitung der Zeile 100 werden folgende Zeilen abgearbeitet:

110, 500, 510, 520, 530, 540, 120, ...

Nach Abarbeitung der Zeile 230 werden folgende Zeilen abgearbeitet:

240, 500, 510, 520, 530, 540, 250, ...

Unterprogrammrufe mit Hilfe von gosub- und on-gosub-Anweisungen koennen beliebig verschachtelt werden.
Es sei nochmals darauf hingewiesen, dass das hier beschriebene Unterprogrammkonzept viele Maengel hat und deshalb nach Moeglichkeit vermieden werden sollte. Die im Abschnitt 9. beschriebenen Ausdrucksmittel bieten einen vollwertigen Ersatz und darueber hinaus eine Reihe weiterer Moeglichkeiten. Das oben angefuehrte Problem ist zweckmaessiger mit Hilfe einer internen Subroutine zu loesen.

8.3. Schleifen

Schleifen werden ueberall dort verwendet, wo bestimmte Programmteile zyklisch durchlaufen werden sollen.
Schleifen koennen als Zaehlschleifen oder als do-Schleifen aufgebaut werden. Zaehlschleifen werden beendet, sobald die Laufvariable den durch Anfangs- und Endwert gegebenen Bereich verlaesst. Bei do-Schleifen wird die Zyklusbeendigung durch Bedingungen gesteuert.

Syntax

```
                  |-                -|
                  | do-schleife      |
schleife ::= <    |                  |  >
                  | zaehlschleife    |
                  |_                _|

                                        *
do_schleife ::= do_zeile [block] loop_zeile

do_zeile ::=
   zeilennummer DO [endebedingung] zeilenrest

loop_zeile ::=
   zeilennummer LOOP [endebedingung] zeilenrest

                       |-      -|
                       | WHILE  |
endebedingung ::= <    |        |  > logischer_ausdruck
                       | UNTIL  |
                       |_      _|

exit_do_anweisung ::= EXIT DO

                                        *
zaehlschleife ::= for_zeile [block] next_zeile

for_zeile ::=
   zeilennummer FOR laufvariable = anfangswert TO endwert
                                   STEP schrittweite zeilenrest

next_zeile ::=
   zeilennummer NEXT laufvariable zeilenrest

laufvariable ::= einfache_num_variable

anfangswert ::= num_ausdruck

endwert ::= num_ausdruck

schrittweite ::= num_ausdruck

exit_for_anweisung ::= EXIT FOR
```

Do-Schleifen werden solange ausgefuehrt, bis eine angegebene Endebedingung erfuellt ist oder bis das Verlassen der do-Schleife durch eine exit-do-Anweisung oder eine andere Steueranweisung erzwungen wird. Eine Endebedingung mit dem Schluesselwort WHILE ist erfuellt, wenn der nachfolgende logische Ausdruck das Ergebnis "falsch" liefert. Eine Endebedingung mit UNTIL ist erfuellt, sobald der logische Ausdruck den Wert "wahr" liefert.
Bei der Abarbeitung der do-Zeile wird zur naechsten Zeile uebergegangen, wenn keine Endebedingung angegeben war, oder wenn diese nicht erfuellt war. Bei erfuellter Endebedingung wird die auf die loop-Zeile folgende Anweisung abgearbeitet.
Bei Erreichen der loop-Zeile wird zurueck zur do-Zeile verzweigt, wenn keine Endebedingung angegeben ist, oder wenn diese nicht erfuellt ist. Sonst wird die nachfolgende Zeile abgearbeitet. Do-Schleifen ohne Endebedingung koennen nur mittels exit-do-Anwei-

sung oder anderen Steueranweisungen (goto-Anweisung) verlassen werden. Die exit-do-Anweisung verzweigt zu der auf die loop-Zeile folgenden Anweisung. Sie darf nur innerhalb von do-Schleifen verwendet werden. Sind mehrere do-Schleifen ineinander verschachtelt, wirkt die exit-do-Anweisung auf die innerste dieser do-Schleifen.
Do-Schleifen duerfen nicht von aussen durch Sprunganweisungen betreten werden.

Beispiel

Das folgende Programmstueck liest ueber das Terminal solange Zeichenketten ein und speichert sie in einem Stringfeld ab, bis eine leere Zeichenkette eingegeben wird, oder bis alle Feldelemente belegt sind.

```
10 PROGRAM instring
20 DIM array□(1 TO 50)
30 LET I = LBOUND(array□)
40   DO WHILE I <= UBOUND(array□)
50   LINE INPUT string□
60   IF string□ = "" THEN EXIT DO
70   LET array□(I) = string□
80 LET I = I+1
90 LOOP
```

Die Abarbeitung einer Zaehlschleife wird durch die folgende Ersatzkonstruktion erklaert:

```
- for-Anweisung
  FOR laufvariable = anfangswert TO endwert STEP schritt
  .
  .
  .
  (Anweisungen)
  .
  .
  .
  NEXT laufvariable

- dazu aequivalente do-Schleife

  LET hilfsvariable1 = endwert
  LET hilfsvariable2 = schritt
  LET laufvariable = anfangswert

  DO UNTIL (laufvariable - hilfsvariable1)*SGN(hilfsvariable2)>0
      .
      .
      .
  (Anweisungen)
      .
      .
      .
  LET laufvariable = laufvariable + hilfsvariable2
  LOOP
```

Jede Zaehlschleife besitzt ihre eigenen Hilfsvariablen. Auf diese Hilfsvariable kann nicht durch andere Anweisungen des Programms zugegriffen werden. Wenn in der for-Zeile keine Schrittweite

angegeben ist, wird die Schrittweite +1 angenommen. Eine Zaehlschleife kann vorzeitig durch eine exit-for-Anweisung oder durch eine Sprunganweisung verlassen werden. Sind mehrere ineinander verschachtelte Zaehlschleifen aktiv, wird durch die exit-for-Anweisung die innerste Zaehlschleife beendet. Eine exit-for-Anweisung darf nur innerhalb einer Zaehlschleife stehen.
Beim Verlassen einer Zaehlschleife durch eine exit-for-Anweisung oder durch eine andere Steueranweisung behaelt die Laufvariable den Wert, den sie vor Ausfuehrung der exit-for-Anweisung bzw. der betreffenden Steueranweisung hatte. Wird die Zaehlschleife an der next-Zeile verlassen, hat die Laufvariable den ersten Wert, der den Endwert ueberschreitet bzw. bei negativer Schrittweite unterschreitet. Bei der Verschachtelung von Zaehlschleifen ist auf die korrekte Folge der next-Zeilen zu achten. Innerhalb verschachtelter Zaehlschleifen darf nicht zweimal die gleiche Laufvariable verwendet werden. In Zaehlschleifen darf nicht von aussen verzweigt werden.

Beispiel fuer korrekte Schachtelung von Zaehlschleifen

```
10 FOR I = 1 TO 10
20   FOR J = 1 TO 5
30     FOR K = 1 TO 20
       .
       .
       .
  (Anweisungen)
       .
       .
       .
120      NEXT K
130    NEXT J
140 NEXT I
```

Beispiel fuer unkorrekte Schachtelung von Zaehlschleifen

```
10 FOR J = -1 TO 3
20 FOR K = J + 1 TO 5
   .
   .
   .
120 NEXT J
130 NEXT K
```

Beispiel fuer die Abarbeitung von Zaehlschleifen.
Es werden Anfangswert, Endwert und Schrittweite eingegeben. Als Ausgabe erscheinen auf dem Terminal alle Werte, die die Laufvariable annimmt und der Wert, den sie nach Verlassen der Zaehlschleife hat.

```
10 PROGRAM Zaehlertest
20 INPUT aw,ew,sw
30 FOR lv = aw TO ew STEP sw
40   PRINT lv
50 NEXT lv
60 PRINT "Wert nach verlassen der Schleife";lv
70 END
```

Die folgende Tabelle enthaelt einige Eingabe- und Ausgabefolgen, die das Programm "Zaehlertest" liefert.

Eingaben			Ausgaben					
aw	ew	sw	Werte der Laufvariablen					Wert nach Verlassen
2	5	1	2	3	4	5		6
2	5	2	2	4				6
1	-3	-1	1	0	-1	-2	-3	-4
1	-3	-1.2	1	-0.2	-1.4	-2.6		-3.8

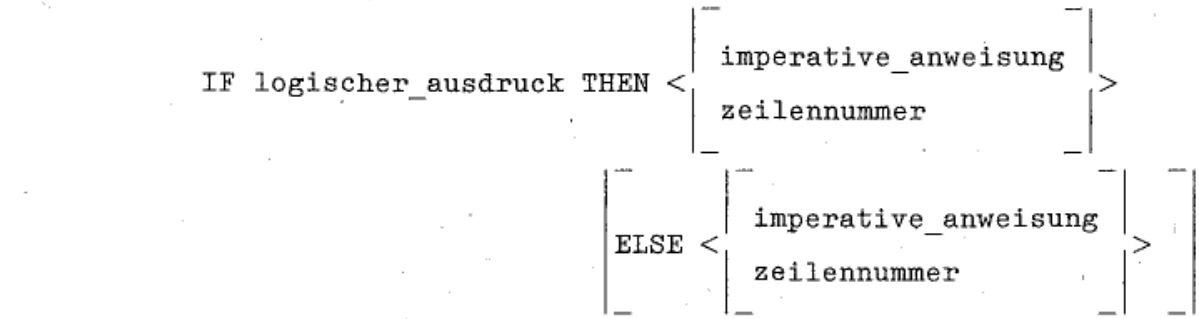
8.4. If-Anweisung und Fallauswahl

BASIC unterscheidet zwei Typen der if-Anweisung.
Mit der einfachen if-Anweisung kann die Abarbeitung einer imperativen Anweisung vom Wert eines logischen Ausdrucks abhaengig gemacht werden bzw. es wird eine von zwei imperativen Anweisungen ausgefuehrt.
Beim if-Block wird die Abarbeitung ganzer Anweisungsfolgen durch logische Ausdruecke gesteuert.
Die Fallauswahl ermoeglicht die Auswahl einer auszufuehrenden Anweisungsfolge in Abhaengigkeit von einem berechneten numerischen- oder String-Wert.

8.4.1. If-Anweisung und If-Block

Syntax

if_anweisung ::=



if_block ::=

```
if_then_zeile
    then_block [elseif_block]* [else_block] end_if_zeile
```

if_then_zeile ::=

```
zeilennummer IF logischer_ausdruck THEN zeilenrest
```

then_block ::= [block]*

```
elseif_block ::= elseif_then_zeile [block]*

elseif_then_zeile ::=

   zeilennummer ELSEIF logischer_ausdruck THEN zeilenrest

else_block ::= else_zeile [block]*

else_zeile ::= zeilennummer ELSE zeilenrest

end_if_zeile ::= zeilennummer END IF zeilenrest
```

Bei einer if-Anweisung wird zuerst der Wert des logischen Ausdrucks berechnet. Ist dieser Wert "wahr", wird die imperative Anweisung nach THEN ausgefuehrt bzw. es wird zu der Zeile verzweigt, deren Nummer nach THEN angegeben ist. Ist der Wert des logischen Ausdrucks "falsch", wird die imperative Anweisung nach ELSE ausgefuehrt bzw. zu der Zeile verzweigt, deren Nummer nach ELSE angegeben ist. Fehlt der ELSE-Teil, wird die naechste Anweisung abgearbeitet.
Bei einem if-Block wird ebenfalls zuerst der Wert des logischen Ausdrucks in der if-then-Zeile berechnet. Liefert dieser den Wert "wahr", so werden die Anweisungen des then-Blockes ausgefuehrt und danach die auf die end-if-Zeile folgende Anweisung. Ist der logische Wert in der if-then-Zeile falsch, wird zur ersten else-if-Zeile verzweigt. Fehlt diese, wird der else-Block abgearbeitet. Wenn auch dieser fehlt, wird zur Anweisung, die auf die end-if-Zeile folgt, verzweigt.
Die Abarbeitung einer elseif-Zeile erfolgt analog zur Abarbeitung der then-Zeile. Zuerst wird der logische Ausdruck nach ELSEIF berechnet. Liefert dieser den Wert "wahr", werden die Anweisungen des elseif-Blockes ausgefuehrt, anschliessend die Anweisung nach der end-if-Zeile. Ergibt er den Wert "falsch", wird zur naechsten elseif-Zeile verzweigt, bzw. zum else-Block, falls keine weitere elseif-Zeile existiert oder zur Anweisung nach der end-if-Zeile, wenn auch der else-Block fehlte.
Es ist zulaessig, den then-Block, den elseif-Block bzw. den else-Block durch Sprunganweisungen oder andere Steueranweisungen zu verlassen. Es ist jedoch nicht moeglich diese Bloecke von aussen durch eine Verzweigung zu betreten. If-Bloecke koennen ineinander geschachtelt werden.

Beispiel

Im folgenden Beispiel wird ueberprueft, ob eine Abfrage korrekt mit "ja" oder "nein" beantwortet wird. Fuer "ja" wird der Steuerwert 1 erzeugt, fuer "nein" 0.
Ist die Antwort weder "ja" noch "nein", wird die Abfrage wiederholt.

Verwendung von if-Anweisungen:

```
250  INPUT antwort$
260  IF antwort$ = "ja" THEN 280
270  IF antwort$ = "nein" THEN 300 ELSE 320
280  LET steuerwert = 1
290  GOTO 340
300  LET steuerwert = 0
310  GOTO 340
320  PRINT "mit ja oder nein antworten"
330  GOTO 250
340  ! Ende Abfrage
```

Verwendung eines if-Blockes:

```
250  INPUT antwort$
260  IF antwort$ = "ja" THEN
270     LET steuerwert = 1
280  ELSEIF antwort$ = "nein" THEN
290     LET steuerwert = 0
300  ELSE
310     PRINT "mit ja oder nein antworten"
320     GOTO 250
330  END IF
```

Es wird deutlich, dass die Verwendung eines if-Blockes an dieser Stelle zu einer uebersichtlicheren Programmstruktur fuehrt.

8.4.2. Fallauswahl

Syntax

```
select_block ::=
    select_zeile[kommentarzeile]* {case_block}* [case_else_block]
                                                  end_select_zeile

select_zeile ::= zeilennummer SELECT CASE ausdruck zeilenrest

case_block ::= case_zeile [block]*

case_else_block ::=  case_else_zeile [block]*

end_select_zeile ::= zeilennummer END SELECT zeilenrest

case_zeile ::= zeilennummer CASE case_liste zeilenrest

case_else_zeile ::= zeilennummer CASE ELSE zeilenrest

case_liste ::= <|case_bedingung|>*
```

```
                    |                        |
                    | konstante              |
case_bedingung ::= <| einseitiger_bereich    |>
                    | zweiseitiger_bereich   |
                    |                        |
```

einseitiger_bereich ::= IS relation konstante

zweiseitiger_bereich ::= konstante TO konstante

relation ::= =|<>|><|<|>|<=|=<|>=|=>

Bei der Abarbeitung eines select-Blockes wird zunaechst der Wert des Ausdrucks in der select-Zeile berechnet. Dieser Wert wird anschliesend mit den Konstanten in den case-Bedingungen der case-Zeilen verglichen.
Liefert eine case-Bedingung den Wert "wahr" werden die Anweisungen des zugehoerigen case-Blockes ausgefuehrt. Danach wird die Anweisung, die auf die end-select-Zeile folgt abgearbeitet.
Liefert keine der case-Bedingungen den Wert "wahr", werden die Anweisungen im case-else-Block ausgefuehrt, falls ein case-else-Teil angegeben war. Sonst entsteht eine nicht triviale Ausnahme.

Wenn als case-Bedingung nur eine Konstante angegeben ist, so liefert diese Bedingung den Wert "wahr", wenn der berechnete Ausdruckswert gleich dem Wert der Konstanten ist. Ist als case-Bedingung ein einseitiger Bereich angegeben, liefert sie den Wert "wahr", wenn der Wert des Ausdrucks in der select-Zeile in der angegbenen Relation zur Konstanten steht. Bei einem zweiseitigem Bereich erhaelt man den Wert "wahr", wenn der Ausdruckswert groesser oder gleich der ersten und kleiner oder gleich der zweiten Konstanten ist.
Die case-Bedingungen in den select-Zeilen muessen so angegeben werden, dass fuer jeden beliebigen Ausdruckswert hoechstens eine case-Bedingung im gesamten case-Block den Wert "wahr" liefert.
Ist in der select-Zeile ein numerischer Ausdruck angegeben, duerfen die case-Bedingungen nur numerische Konstanten enthalten.
Enthaelt die select-Zeile dagegen einen string-Ausdruck, dann sind in den case-Bedingungen ausschliesslich string-Konstanten zulaessig.
case-Bloecke und case-else-Bloecke duerfen durch Sprunganweisungen verlassen werden. Sie duerfen jedoch nicht von aussen durch Sprunganweisungen betreten werden.

Beispiel

Im folgenden Programmteil wird in einer Fallauswahl das Porto fuer einen Brief im Fernverkehr berechnet.

```
300  SELECT CASE Round (gewicht,0)
310  !
320  ! Gewicht in Gramm, ganzzahlig gerundet
330  ! Porto in Pfennigen
340  !
350  CASE IS <= 20
360     LET porto = 20
370  CASE 21 TO 250
380     LET porto = 40
390  CASE 251 TO 500
400     LET porto = 60
410  CASE ELSE
420     PRINT "Brief wiegt ueber 500 g !"
430  END SELECT
```

9. Unterprogrammtechnik und Programmverkettung

BASIC bietet mit der Unterprogrammtechnik und der Moeglichkeit der Programmverkettung geeignete Mittel zu einer uebersichtlichen Einteilung des Programms in funktionell eigenstaendige Einheiten. Der Benutzer kann:

- eigene Funktionen definieren, die wie Standardfunktionen gerufen werden und deren Funktionswert in numerischen oder string-Ausdruecken verwendet werden kann,
- Subroutinen schreiben, die mit der call-Anweisung gerufen werden und die ueber Parameter Werte uebernehmen und uebergeben koennen,
- mehrere Hauptprogramme durch chain-Anweisungen so miteinander verketten, dass sie nacheinander ohne Bedienereingriffe ablaufen.

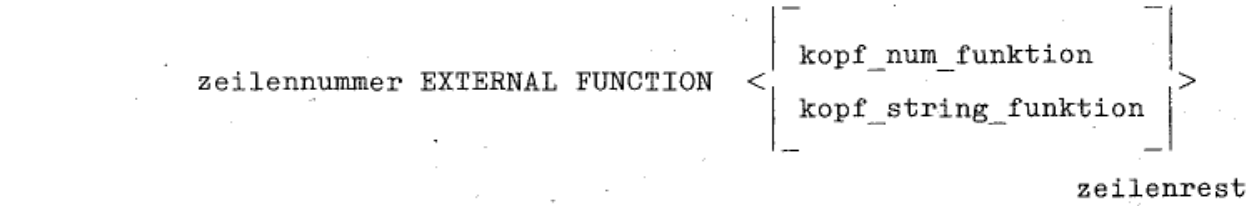
9.1. Funktionen

9.1.1. Funktionsdefinition

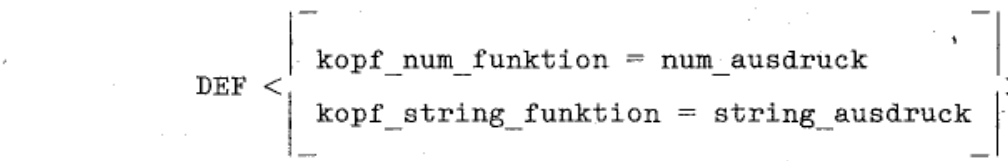
Syntax

```
funktionsdefinition ::= <[ interne_funktionsdefinition      ]>
                        [ externe_funktionsdefinition       ]

interne_funktionsdefinition ::=
<[ zeilennummer  def_anweisung  zeilenrest                  ]>
 [ internal_function_zeile {block}* end_function_zeile      ]

externe_funktionsdefinition ::=
   external_function_zeile <[ block              ]>*
                           [ interne_prozedur    ]
                                        end_function_zeile

internal_function_zeile ::=
   zeilennummer FUNCTION <[ kopf_num_funktion    ]> zeilenrest
                         [ kopf_string_funktion  ]
```

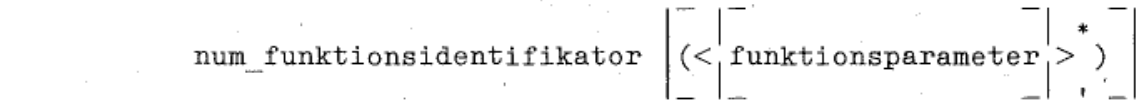
external_function_zeile ::=



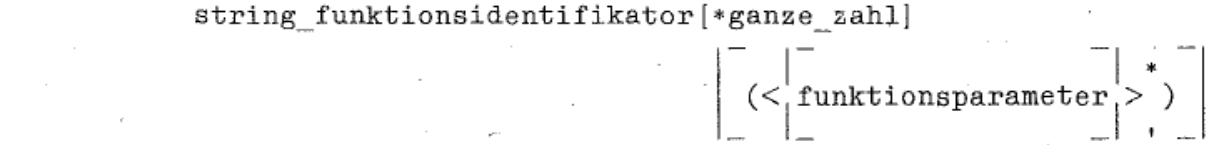
def_anweisung ::=



kopf_num_funktion ::=



kopf_string_funktion ::=



num_funktionsidentifikator ::= num_identifikator

string_funktionsidentifikator ::= string_identifikator

end_function_zeile ::= END FUNCTION

Funktionen koennen entweder als externe oder als interne Funktionen definiert werden. Externe Funktionen sind im gesamten Programm gueltig. Der Name einer externen Funktion muss sich von den Namen der uebrigen externen Funktionen und Subroutinen des Programms sowie vom Namen des Hauptprogramms unterscheiden.
Interne Funktionen gelten in der Programmeinheit, in der sie definiert sind. Der Name einer internen Funktion darf innerhalb der entsprechenden Programmeinheit nur zur Bezeichnung dieser internen Funktion verwendet werden.
Eine Funktion darf weder durch Sprunganweisungen verlassen, noch von aussen durch Sprunganweisungen betreten werden. Verzweigungen innerhalb einer Funktion duerfen nicht zur external-function-Zeile bzw. internal-function-Zeile dieser Funktion fuehren. Wird bei der Abarbeitung eines Programms eine interne Funktionsdefinition erreicht, so wird diese Funktionsdefinition ohne Wirkung uebergangen, d.h. es wird zu der Zeile verzweigt, die auf die interne Funktionsdefinition folgt.

9.1.2. Spezifikation von Funktionen

Vor der ersten Benutzung einer externen Funktion muss die externe Funktion durch eine declare-Anweisung spezifiziert werden. Vor der ersten Benutzung einer internen Funktion muss entweder die interne Funktion definiert worden sein, oder sie muss ebenfalls durch eine declare-Anweisung spezifiziert werden. Direkt rekursive Rufe sind in beiden Faellen ohne vorherige Spezifikation zulaessig.

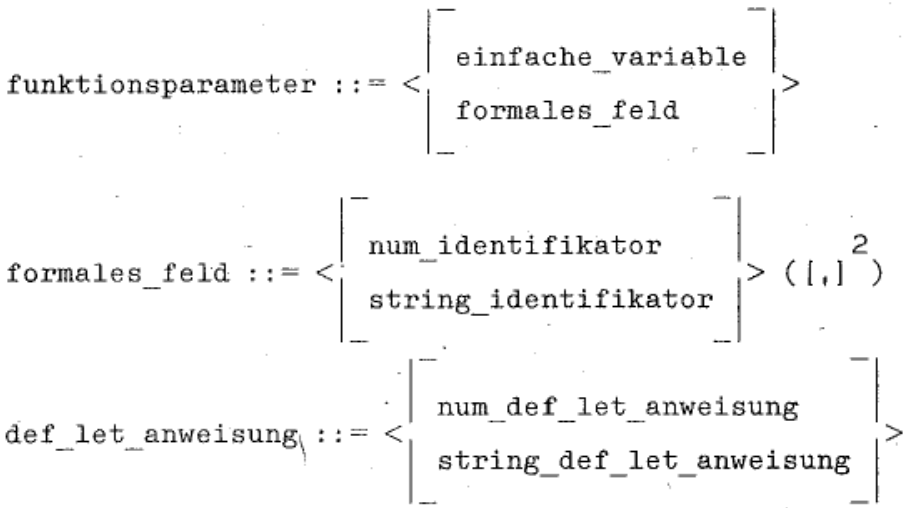
Syntax

```
declare_anweisung ::= DECLARE typ_deklaration

                        ┌                        ┐
                        │ external_function_typ  │
typ_deklaration ::= <   │ internal_function_typ  │> funktionsliste
                        │ def_typ                │
                        └                        ┘

external_function_typ ::= EXTERNAL FUNCTION

internal_function_typ ::= INTERNAL FUNCTION

def_typ ::= DEF
                        ┌                        ┐ *
funktionsliste ::= <    │funktionsidentifikator  │>
                        └                        ┘ ,

                                ┌                        ┐
                                │ num_identifikator      │
funktionsidentifikator ::= <    │                        │>
                                │ string_identifikator   │
                                └                        ┘
```

Durch eine declare-Anweisung mit einem external-function-Typ werden die in der Funktionsliste aufgefuehrten Identifikatoren als Namen externer Funktionen spezifiziert. Nach der Spezifikation koennen die Funktionen in numerischen- bzw. String-Ausdruecken verwendet werden. Die spezifizierten externen Funktionen muessen im Programm definiert werden. Durch eine declare-Anweisung mit einem internal-function-Typ oder einem def-typ werden die in der Funktionsliste aufgefuehrten Identifikatoren als Namen interner Funktionen spezifiziert. Die spezifizierten internen Funktionen muessen in der gleichen Programmeinheit definiert werden.
Eine in der betreffenden Programmeinheit definierte interne Funktion darf in dieser Programmeinheit nicht mehr spezifiziert werden. Eine mit dem def-Typ spezifizierte interne Funktion muss durch eine def-Anweisung definiert werden, eine mit dem internal-function-Typ spezifizierte Funktion muss mit einer internal-function-Zeile beginnen. Die spezifizierten Funktionsnamen duerfen in der betreffenden Progammeinheit keine andere Bedeutung besitzen.
Ein Funktionsname darf nur einmal in einer Funktionsliste auftreten. Eine Funktionsspezifikation gilt in der Programmeinheit, in der sie auftritt. Wird eine externe Funktion in mehreren Programmeinheiten benutzt, muss sie in jeder dieser Programmeinheiten erneut spezifiziert werden.

9.1.3. Parameteruebergabe, Berechnung des Funktionswertes

Syntax



num_def_let_anweisung ::=

LET num_funktionsidentifikator = num_ausdruck

string_def_let_anweisung ::=

LET string_funktionsidentifikator = string_ausdruck

exit_function_anweisung ::= EXIT FUNCTION

Beim Aufruf einer Funktion werden die Werte der Funktionsargumente berechnet und den Funktionsparametern zugewiesen. Bei der Uebergabe von numerischen Werten muessen daher die gerufene externe Funktion und die rufende Programmeinheit die gleiche Arithmetik-Option haben. Bei der Uebergabe von string-Feldern muss die gerufene externe Funktion ebenfalls die gleiche Arithmetik-Option haben, wie die Programmeinheit, aus der der Ruf erfolgte, da bei Feldern die Indexgrenzen, die aktuelle und maximale Feldgroesse u.a. numerische Werte mit uebermittelt werden muessen.
Die Parameteruebergabe erfolgt "by value", d.h., dass der Wert von Funktionsargumenten nicht durch eventuelle Wertzuweisungen an den entsprechenden Funktionsparameter bei der Abarbeitung der Funktion veraendert werden kann. Bei formalen Feldern wird dazu eine Kopie des Funktionsargumentes bereitgestellt.
Die Zahl und der Typ der Funktionsargumente muessen mit der Zahl und dem Typ der Funktionsparameter uebereinstimmen.
Die Zahl der Dimensionen eines Feldargumentes muss mit der Zahl der Dimensionen des Feldparameters uebereinstimmen.
Der Feldparameter erhaelt beim Funktionsaufruf die aktuellen Indexgrenzen des Feldargumentes. Stringparameter duerfen als Werte Zeichenketten zwischen 0 und 32767 Zeichen haben.
Die Identifikatoren der Parameter duerfen in der Liste der Funktionsparameter nur einmal auftreten.
Die Parameter einer externen Funktion sind in der gesamten Programmeinheit gueltig und duerfen nicht zur Bezeichnung anderer

Objekte verwendet werden. Die Parameter interner Funktionen gelten nur in der internen Funktion.
Interne Funktionen koennen auf Objekte der umgebenden Programmeinheit zugreifen, deren Werte veraendern bzw. Operationen mit Kanaelen der umgebenden Programmeinheit ausfuehren, d.h. die Objekte einer Programmeinheit sind zu den darin enthaltenen internen Funktionen global.
Die Objekte einer externen Funktion (Variable, Felder, Daten aus data-Anweisungen und Kanaele ausser dem Terminalkanal) sind zu jedem Aufruf dieser Funktion lokal. Beim Aufruf der externen Funktion erhalten deren Variablen und Felder initiale Werte (s. Abschn., 5.1. und 6.1.), beim Verlasssen der externen Funktion gehen die Werte der Variablen und Felder verloren bzw. offene Kanaele werden geschlossen.
Sind bei rekursiven externen Funktionen gleichzeitig mehrere Aufrufe ein und derselben Funktion aktiv, so besitzt jeder der aktiven Funktionsrufe seine eigenen Variablen und Felder usw.
Bei internen Funktionen, die mit der def-Anweisung definiert wurden, wird der Funktionswert durch Abarbeitung des rechts vom Gleichheitszeichen stehenden Ausdruckes ermittelt. Bei allen anderen Funktionen muss der Funktionswert durch eine def-let-Anweisung zugewiesen werden. Jede dieser Funktionen muss mindestens eine def-let-Anweisung enthalten. Als Zielvariable steht in der def-let-Anweisung links vom Gleichheitszeichen der Name der Funktion. Der Ausdruck rechts vom Gleichheitszeichen liefert den Funktionswert. Wird bei der Abarbeitung einer Funktion keine der darin enthaltenen def-let-Anweisungen ausgefuehrt, bleibt der Funktionswert undefiniert. Die Zuweisung des Funktionswertes ist nur durch eine def-let-Anweisung moeglich, der Funktionsname kann nicht in anderen Anweisungen (input-Anweisung, let-Anweisung mit mehreren Zielvariablen) als Ziel von Wertzuweisungen fungieren.
Bei einer string_def_let_Anweisung ist zu beachten, dass der Funktionswert eine eventuell nach dem Identifikator der Stringfunktion im Kopf der Funktionsdeklaration angegebene Laengenbegrenzung nicht ueberschreitet. Anderenfalls wird eine nicht triviale Ausnahme ausgeloest.
Eine Funktion wird beim Erreichen der end-function-Zeile verlassen. Soll eine Funktion an einer anderen Stelle verlassen werden, kann man dazu die exit-funktion-Anweisung verwenden.
Die exit-function-Anweisung ist nur innerhalb von Funktionen zulaessig.

Beispiel

Im folgenden Beispiel werden zwei Funktionen definiert, die eine Zeichenfolge zur Bildschirmsteuerung liefern. Die Funktion CLEAR□ liefert eine Zeichenfolge zum Bildschirmloeschen, die Funktion CURSOR□ liefert eine Zeichenfolge zur Kursorpositionierung. Beide Funktionen werden im Hauptprogramm deklariert und anschliessend als externe Funktion definiert.

```
10  PROGRAM Bildschirmsteuerung
20  ! Deklaration der benutzten Funktionen
30  DECLARE EXTERNAL FUNCTION Cursor□, Clear□
40  ! Bildschirm loeschen und Ausgabe von "***"
45  ! ab Zeile 12, Spalte 30
50  PRINT Clear□; Cursor□(12,30); "***"
60  END
70  !
80  ! Funktion Clear□
90  !
100 EXTERNAL FUNCTION Clear□
110 LET Clear□ = CHR□(27)&CHR□(91)&"2J"
120 END FUNCTION
130 !
140 ! Funktion Cursor□
150 !
160 EXTERNAL FUNCTION Cursor□ (Zeile, Spalte)
170 LET CURSOR□ = CHR□(27)&CHR□(91)&STR□(Zeile)          &
&    & ";"&STR□(Spalte)&"H"
180 END FUNCTION
```

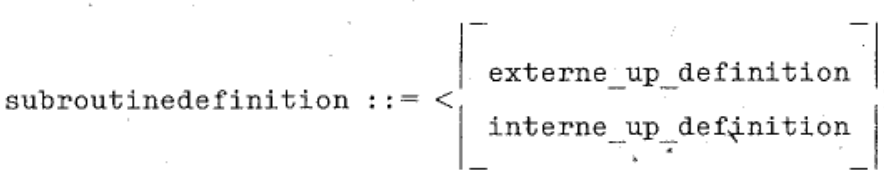
Im folgenden Programm wurde die interne Funktion Pow2 zur Berechnung der Zweierpotenz nicht deklariert. Das ist moeglich, da der Ruf der Funktion auf Zeile 70 ein direkt rekursiver Ruf ist und dem Ruf auf Zeile 110 die Definition der internen Funktion vorangeht.

```
10  PROGRAM Zweierpotenz
20  FUNCTION Pow2 (N)
40  IF N = 0 THEN
50     LET Pow2 = 1
60  ELSE
70     LET Pow2 = 2 * Pow2 (N-1)
80  END IF
90  END FUNCTION
100  FOR I = 1 TO 10 ! 2^1 bis 2^10 ausgeben
110    PRINT I, Pow2 (I)
120  NEXT I
130  END
```

9.2. Subroutinen

9.2.1. Subroutinedefinition

Syntax



```
externe_up_definition ::=
                             _                  _  *
                            |  block             |
    external_sub_zeile   <  |                    |>  end_sub_zeile
                            |  interne_prozedur  |
                             -                  -
interne_up_definition ::=
                                      *
    internal_sub_zeile {block} end_sub_zeile

external_sub_zeile ::=

    zeilennummer EXTERNAL SUB up_kopf zeilenrest

internal_sub_zeile ::=

    zeilennummer SUB up_kopf zeilenrest
                                        _    _                  _    _
                                       |    |                  | *  |
up_kopf ::= routine_identifikator      | (< | routineparameter >  ) |
                                       |    |                  | ,  |
                                        -    -                  -    -
end_sub_zeile ::= END SUB
```

Subroutinen koennen als externe oder als interne Subroutinen definiert werden. Externe Subroutinen sind im gesamten Programm gueltig. Interne Subroutinen gelten in der Programmeinheit, in der sie definiert sind. Der Name einer externen Subroutine muss sich von den Namen der uebrigen externen Subroutinen und externen Funktionen und vom Namen des Hauptprogramms unterscheiden. Der Name einer internen Subroutine darf in der betreffenden Programmeinheit nur zur Bezeichnung dieser internen Subroutine verwendet werden.
Eine Subroutine darf weder durch Sprunganweisungen verlassen, noch von aussen durch Sprunganweisungen betreten werden. Aus einer Subroutine heraus darf nicht zur external-sub-Zeile bzw. zur internal-sub-Zeile verzweigt werden.
Wird bei der Abarbeitung eines Programms eine interne_up_Definition erreicht, so wird diese ohne Wirkung uebergangen, d.h. es wird zu der Zeile verzweigt, die auf die interne-up-Definition folgt.

9.2.2. Spezifikation von Subroutinen

Externe Subroutinen muessen vor ihrer ersten Benutzung spezifiziert werden. Interne Subroutinen muessen dann spezifiziert werden, wenn der erste Aufruf der internen Subroutine durch eine call-Anweisung vor der Definition der internen Subroutine erfolgt. Direkt rekursive Rufe sind in beiden Faellen ohne vorherige Spezifikation zulaessig. Die Spezifikation von Subroutinen erfolgt in einer declare-Anweisung.

Syntax

```
declare_anweisung ::= DECLARE typ_deklaration

                      ┌                    ┐
                      │ external_sub_typ   │
typ_deklaration ::= <│                    │> up_liste
                      │ internal_sub_typ   │
                      └                    ┘

external_sub_typ ::= EXTERNAL SUB

internal_sub_typ ::= INTERNAL SUB

                 ┌                    ┐*
up_liste ::= <│routine_identifikator│>
                 └                    ┘,
```

Durch eine declare-Anweisung mit einem external-sub-Typ werden die in der up-Liste aufgefuehrten Routineidentifikatoren als Namen externer Subroutinen spezifiziert.
Die spezifizierten externen Subroutinen muessen im Programm definiert werden. Durch eine declare-Anweisung mit einem internal-sub-Typ werden die in der up-Liste aufgefuehrten Routineidentifikatoren als Namen interner Subroutinen definiert. Die spezifizierten internen Subroutinen muessen in der gleichen Programmeinheit definiert werden. Eine in der betreffenden Programmeinheit bereits definierte interne Subroutine darf in dieser Programmeinheit nicht mehr spezifiziert werden.
Nach der Spezifikation koennen die spezifizierten Subroutinen durch call-Anweisungen aufgerufen werden. Die spezifizierten Namen der Subroutinen duerfen in der betreffenden Programmeinheit keine andere Bedeutung besitzen.
Eine Subroutine darf in einer Programmeinheit nur einmal in einer up-Liste spezifiziert werden. Die Spezifikation gilt fuer die Programmeinheit, in der sie auftritt.
Wird eine externe Subroutine in mehreren Programmeinheiten benutzt, muss sie in jeder dieser Programmeinheiten erneut spezifiziert werden.

9.2.3. Parameteruebergabe, CALL-Anweisung

Syntax

```
                      ┌                    ┐
                      │ einfache_variable  │
routineparameter ::= <│ formales_feld      │>
                      │ formaler_kanal     │
                      └                    ┘

                    ┌                    ┐
                    │ num_identifikator  │
formales_feld ::= <│                    │> ([,]^2)
                    │ string_identifikator│
                    └                    ┘

formaler_kanal ::= *ganze_zahl
```

```
call_anweisung ::= CALL routine_identifikator [(< up_argument >)]*

up_argument ::= < by_value_argument
                  by_reference_argument
                  kanal_argument >

by_reference_argument ::= < variable
                            feld >

by_value_argument ::= < num_ausdruck
                        string_ausdruck >

kanal_argument ::= #num_ausdruck

exit_sub_anweisung ::= EXIT SUB
```

Bei der Ausfuehrung einer call-Anweisung werden die up-Argumente an die Routineparameter uebergeben. Bei der Uebergabe von numerischen Werten und von Feldern (einschliesslich Stringfelder) muessen die gerufene externe Subroutine und die rufende Programmeinheit die gleiche Arithmetik-Option haben. Die Parameteruebergabe erfolgt fuer Felder, fuer einfache Variable, fuer indizierte Variable und fuer Kanaele "by reference", fuer Konstanten, fuer Werte aus der Berechnung von Ausdruecken und fuer Stringvariable mit Teilstring-Angabe erfolgt sie "by value".
Bei der Parameteruebergabe "by value" erhaelt der Routineparameter der Subroutine eine Kopie des zu uebergebenden up-Argumentes als Wert. Der Wert des up-Argumentes selbst kann nicht durch Wertzuweisungen an den entsprechenden Routineparameter geaendert werden.
Die Parameteruebergabe "by reference" erfolgt so, dass anschliessend Routineparameter und up-Argument ein und dasselbe Objekt bezeichnen. Die Subroutine kann in diesem Fall den Wert des up-Argumentes bzw. bei Feldern den Wert von Elementen aendern. Die Aenderungen sind in der Programmeinheit, aus der der Aufruf erfolgte, wirksam. Sollen einfache bzw. indizierte Variable "by value" uebergeben werden, schliesst man sie in Klammern ein und bildet so einen Ausdruck.

Beispiel

```
10 PROGRAM para
20 LET a,b = 1
30 SUB assign (par1, par2)
40 LET par1, par2 = 2
50 END SUB
60 CALL assign (a, (b))
70 PRINT a, b
80 END
```

Bei der Abarbeitung dieses Beispiels werden die Zahlen 2 und 1 ausgegeben, da die Variable a in Zeile 60 "by reference" uebergeben wird, so dass sich ihr Wert durch die Zuweisung auf Zeile 40 aendert. Die Variable b behaelt ihren Wert, da sie "by value" uebergeben wurde.

Wenn einer Subroutine ein Feld und gleichzeitig Elemente dieses Feldes "by reference" uebergeben werden, sollte die Subroutine das Feld auf keinen Fall redimensionieren, da das zu undefinierten Aktionen bei der Abarbeitung der Subroutine fuehren kann.
Ein an eine Subroutine uebergebener Kanal muss nicht unbedingt mit einer Datei verbunden sein. Die Subroutine kann den betreffenden Kanal durch eine open-Anweisung mit einer Datei verbinden. Diese Verbindung wird durch das Verlassen der Subroutine nicht beendet.
Der Wert eines Stringparameters kann eine Laenge zwischen 0 und 32767 Zeichen annehmen, wenn die Parameteruebergabe "by value" erfolgte. Wird ein Stringparameter "by reference" uebergeben, so darf der Wert des Stringparameters bei Wertzuweisungen die fuer das Stringargument gueltige Maximallaenge (s. Abschn. 6.5.) nicht ueberschreiten.
Die Zahl und der Typ der up-Argumente muss mit der Zahl und dem Typ der Routineparameter uebereinstimmen. Die Zahl der Dimensionen eines Feldargumentes muss mit der Zahl der Dimensionen des Feldparameters uebereinstimmen. Die Identifikatoren der Parameter duerfen in der Liste der Routineparameter nur einmal auftreten.
Die Parameter einer externen Subroutine sind in der gesamten Programmeinheit gueltig und duerfen dort nicht zur Bezeichnung anderer Objekte verwendet werden. Die Parameter interner Subroutinen gelten nur in der internen Subroutine.
Interne Subroutinen koennen auf Objekte der umgebenden Programmeinheit zugreifen, deren Werte veraendern bzw. Operationen mit Kanaelen der umgebenden Programmeinheit ausfuehren, d.h. die Objekte einer Programmeinheit sind zu den darin enthaltenen internen Subroutinen global.
Die Objekte einer externen Subroutine (Variable, Felder, Daten aus Data-Anweisungen und Kanaele, ausser dem Terminalkanal) sind zu jedem Ruf der externen Subroutine lokal. Beim Aufruf einer externen Subroutine erhalten deren Variable und Felder initiale Werte (s. Abschn. 5.1. und 6.1.). Beim Verlassen der externen Subroutinen gehen die Werte der Variablen und Felder verloren. Offene Kanaele werden geschlossen. Sind bei rekursiven externen Subroutinen gleichzeitig mehrere Rufe ein und derselben Subroutine aktiv, so besitzt jeder der aktiven Rufe seine eigenen Variablen, Felder, Kanaele und seine eigene interne Datenliste.
Die Abarbeitung einer Subroutine beginnt nach der call-Anweisung mit der Zeile, die auf die external-sub-Zeile bzw. die internal-sub-Zeile folgt. Entsprechend den in Abschnitt 4.3. angegebenen Regeln werden die weiteren Anweisungen ausgefuehrt, bis die Subroutine an der end-sub-Zeile oder durch eine exit-sub-Anweisung beendet wird.
Eine exit-sub-Anweisung darf nur innerhalb einer Subroutine stehen.

Beispiel

Mit der folgenden externen Subroutine koennen die Elemente eines eindimensionalen Feldes in aufsteigender Reihenfolge sortiert werden.

```
100 EXTERNAL SUB sort (vektor())
110 DECLARE INTERNAL SUB vertausche
120 LET erster_vektorindex = LBOUND(vektor, 1)
130 LET letzter_vektorindex = UBOUND(vektor, 1)
140 FOR i = erster_vektorindex TO letzter_vektorindex -1
150   FOR j = i+1 TO letzter_vektorindex
160     IF vektor (i) > vektor (j) THEN
170       CALL vertausche (vektor (i), vektor (j))
175     END IF
180   NEXT j
190 NEXT i
200 EXIT SUB
210 SUB vertausche (p1, p2)
220 LET hilfsvariable = p1
230 LET p1 = p2
240 LET p2 = hilfsvariable
250 END SUB
260 END SUB
```

9.3. Programmverkettung

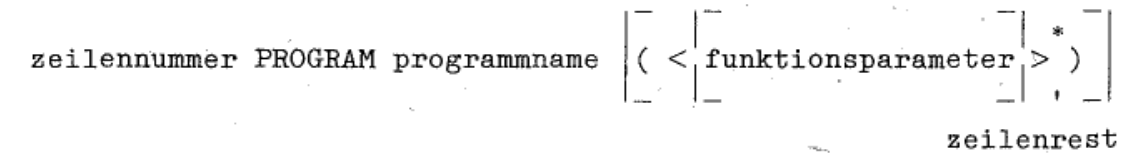
Mit der chain-Anweisung werden Hauptprogramme so miteinander verbunden, dass sie ohne Bedienereingriff nacheinander ablaufen. Die Programmverkettung kann auf diese Weise zum Aufbau grosser Programme angewendet werden.

Syntax

chain-anweisung ::=

programmbeschreiber ::= string_ausdruck

programmzeile ::=



Bei der Abarbeitung einer chain-Anweisung wird das laufende Programm beendet und dafuer das durch den Programmbeschreiber bezeichnete Programm gestartet. Die Funktionsargumente in der chain-Anweisung werden an die Funktionsparameter des zu startenden Programms uebergeben. Fuer diese Parameteruebergabe gelten dieselben Regeln wie fuer die Parameteruebergabe an externe Funktionen (s. Abschn. 9.1.3.). Durch die chain-Anweisung werden noch offene Kanaele des laufenden Programms geschlossen.
Der Programmbeschreiber liefert den Namen des zu startenden Programms (siehe Anleitung fuer Bediener).

In folgenden Faellen verursacht die chain-Anweisung nicht triviale Ausnahmen:

- Das durch den Programmbeschreiber bezeichnete Programm ist nicht verfuegbar.
- Die Zahl der Funktionsargumente in der chain-Anweisung entspricht nicht der Zahl der Funktionsparameter in der Programmzeile des zu startenden Programms.
- Der Typ eines Funktionsargumentes in der chain-Anweisung entspricht nicht dem Typ des entsprechenden Funktionsparameters in der Programmzeile.
- Ein als Funktionsargument angegebenes aktuelles Feld hat nicht die durch den entsprechenden Funktionsparameter geforderte Anzahl der Dimension.
- Zwischen zwei Programmen mit unterschiedlicher Arithmetik-Option werden numerische Parameter oder Stringfelder ueberge-ben.

10. Ein- und Ausgaben

10.1. Eingabe aus der internen Datenliste

In einem Programm koennen durch data-Anweisungen interne Daten vereinbart werden. Diese internen Daten bilden eine zur jeweiligen Programmeinheit lokale interne Datenliste. Mit der read-Anweisung werden Daten aus dieser Datenliste gelesen. Die restore-Anweisung gestattet das mehrfache Lesen von Daten aus der Datenliste.

10.1.1. Data-Anweisung

Syntax

```
                                  ┌                        ┐*
                                  │ konstante              │
data_anweisung ::= DATA <         │                        │>
                                  │ offene_zeichenreihe    │
                                  └                        ┘,

                    ┌                        ┐
                    │ numerische_konstante   │
konstante ::= <     │                        │>
                    │ string_konstante       │
                    └                        ┘

                                  ┌                                        ┐
                                  │ zeichen_1                              │
offene_zeichenreihe ::= <         │                        *               │>
                                  │ zeichen_1 [zeichen_2] zeichen_1        │
                                  └                                        ┘

zeichen_1 ::= buchstabe | ziffer | . | + | -

                    ┌                ┐
                    │ zeichen_1      │
zeichen_2 ::= <     │                │>
                    │ leerzeichen    │
                    └                ┘
```

Die Konstanten und offenen Zeichenreihen in den DATA-Anweisungen einer Programmeinheit werden in der Reihenfolge, wie sie im Programmtext auftreten zur internen Datenliste dieser Programmeinheit verkettet. Durch Stringkonstanten und offene Zeichenreihen werden Stringwerte in der Datenliste dargestellt, numerische Konstanten koennen als numerischer Wert oder als Stringwert verarbeitet werden. Bei der Notation von Stringkonstanten und offenen Zeichenreihen ist der jeweilige Zeichenvorrat zu beachten (siehe Syntax und Abschnitt 6.1.). Ausserdem ist zu beachten, dass Leerzeichen nach DATA bzw. nach oder vor einem Komma nicht zu einer offenen Zeichenreihe gehoeren. Diese beginnt und endet stets mit einem nichtleeren Zeichen (siehe Syntax). Data-Anweisungen werden hauptsaechlich verwendet, um Werte zur Initialisierung von Feldern bereitzustellen.

10.1.2. Read-Anweisung

Syntax

```
read_anweisung ::= READ [missing_behandlung:] variablenliste

                    |        |*
variablenliste ::= <|variable|>
                    |        |,

missing_behandlung ::= IF MISSING THEN ea_fehlerbehandlung

                        |                    |
                        | exit_do_anweisung  |
ea_fehlerbehandlung ::= <| exit_for_anweisung |>
                        | zeilennummer       |
                        |                    |
```

Mit der read-Anweisung werden Daten aus der internen Datenliste der jeweiligen Programmeinheit gelesen und den Variablen der Variablenliste der Reihe nach zugewiesen. Bei indizierten Variablen und bei Stringvariablen mit Teilstringangabe werden deren Indizes erst berechnet, nachdem den in der Variablenliste weiter links stehenden Variablen Werte zugewiesen wurden. Zur Bestimmung des naechsten einzulesenden Datenelementes wird in der Datenliste ein Zeiger gefuehrt. Dieser Zeiger ist zu jedem Aufruf der Programmeinheit lokal und wird bei Aktivierung der Programmeinheit auf das erste Datenelement der ersten data-Anweisung in der Programmeinheit gestellt.
Wird durch eine read-Anweisung ein Wert aus der internen Datenliste verlangt, so wird der Wert gelesen, auf den der Zeiger verweist. Anschliessend wird der Zeiger auf das naechste Element der Datenliste gestellt.
Soll durch eine read-Anweisung ein Element der Datenliste gelesen werden, nachdem bereits alle Elemente verarbeitet worden sind, haengt die weitere Abarbeitung davon ab, ob eine missing-Behandlung angegeben war. Fehlt die missing-Behandlung, wird eine nicht triviale Ausnahme ausgeloest.
War eine missing-Behandlung angegeben, so wird die entsprechende EA-Fehlerbehandlung ausgefuehrt. Dabei wird die angegebene exit-do- oder exit-for-Anweisung ausgefuehrt oder es wird zu der angegebenen Zeilennummer verzweigt (eine exit-do-Anweisung ist nur in einem do-Block, eine exit-for-Anweisung nur in einem for-Block zulaessig).
Nicht triviale Ausnahmen werden auch dann ausgeloest wenn:

- versucht wird fuer eine numerische Variable einen Wert einzulesen, der keine numerische Konstante darstellt,
- versucht wird einer Stringvariablen einen Wert zuzuweisen, dessen Laenge die Maximallaenge dieser Stringvariablen ueberschreitet (s. Abschn. 6.6.),
- versucht wird einer numerischen Variablen einen numerischen Wert zuzuweisen, der nicht im zulaessigen Wertebereich liegt (s. Abschn. 5.).

10.1.3. Restore-Anweisung

Syntax

```
restore_anweisung ::= RESTORE [zeilennummer]
```

Die restore-Anweisung ohne Zeilennummer setzt den Zeiger auf das erste Datenelement der ersten data-Anweisung in der Programmeinheit. Die gesamte Datenliste kann erneut verarbeitet werden.
Ist in der restore-Anweisung eine Zeilennummer angegeben, so wird der Zeiger auf das erste Datenelement dieser Zeile gestellt. Die Zeilennummer muss zu einer Zeile mit einer data-Anweisung in der gleichen Programmeinheit gehoeren. Auf diese Weise koennen zu verarbeitende Teile einer Datenliste ausgewaehlt werden.

Beispiele

Das folgende Programmstueck initialisiert die Elemente des Feldes "Monat□" mit den Werten "Januar" ... "Dezember" und das Feld "Tage" mit der Zahl der Tage je Monat (ohne Beruecksichtigung des Schaltjahres).

```
110 DIM Monat□(1 TO 12), Tage(1 TO 12)
 .
 .
 .
200 DATA  31,28,31       ! 1. Quartal
210 DATA  30,31,30       ! 2. Quartal
220 DATA  31,31,30       ! 3. Quartal
230 DATA  31,30,31       ! 4. Quartal
240 DATA  Januar, Februar, Maerz, April
250 DATA  Mai, Juni, Juli, August, &
& September, Oktober, November, Dezember
 .
 .
 .
300 RESTORE 240
310 FOR i = 1 TO 12
320   READ Monat□(i)
330 NEXT i
 .
 .
 .
400 RESTORE 200
410 FOR i = 1 TO 12
420   READ Tage(i)
430 NEXT i
```

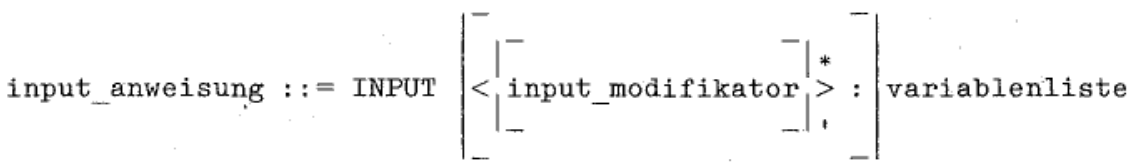
Im folgenden Programmteil werden einzelne Elemente des zweidimensionalen Feldes "Statistik" neu initialisiert. Eine data-Anweisung enthaelt jeweils den Zeilenindex, den Spaltenindex und den Wert fuer ein Feldelement. Die Initialisierung endet, wenn die Datenliste vollstaendig verarbeitet wurde.

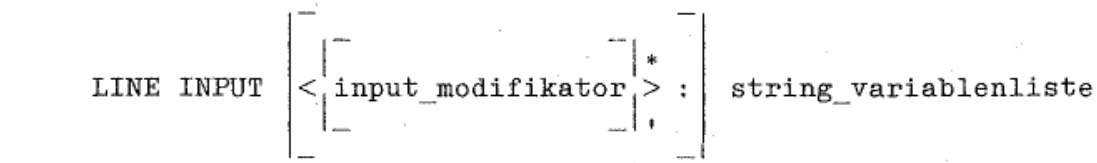
```
.
.
.
215 RESTORE    ! Fuer wiederholtes abarbeiten
220 DO
225 READ IF MISSING THEN EXIT DO:  i,j, statistik(i,j)
230 LOOP
235 !  Daten  Zeile, Spalte, Wert
240 DATA         1 ,      5 ,   19   ! statistik (1,5):=19
250 DATA         3 ,      2 ,   28
255 DATA         3 ,      3,    34
260 DATA         3 ,      4,   -16.2
```

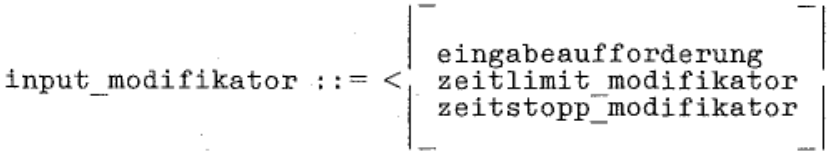
10.2. Eingabe vom Terminal

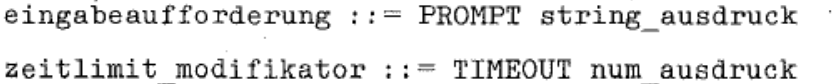
Mit der input-Anweisung und der line-input-Anweisung koennen waehrend des Programmlaufs Werte ueber das Terminal eingegeben und an Variable zugewiesen werden. Die input-Anweisung erlaubt die Eingabe von numerischen- und String-Werten, die durch Kommata getrennt werden. Die line-input-Anweisung ermoeglicht die Eingabe ganzer Zeilen.

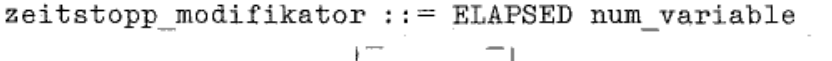
Syntax

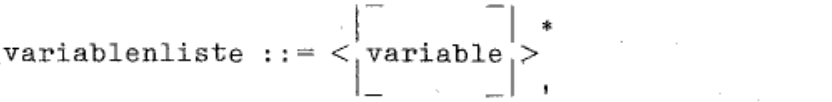


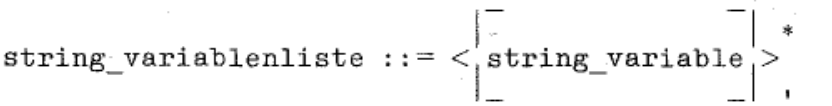












Bei der Ausfuehrung einer input-Anweisung wird ueber das Terminal zunaechst eine Eingabeanforderung ausgegeben. Ist dafuer ein entsprechender input-Modifikator angegeben, erscheint als Eingabeanforderung der Wert des string-Ausdrucks nach dem Schluesselwort PROMPT, sonst wird zur Eingabe mit "? " aufgefordert. Danach wird die Abarbeitung des Programms unterbrochen, bis ueber das Terminal eine Eingabezeile eingegeben wird.

```
input_eingabezeile ::= eingabezeile [,] eol

                      |_          _|  *
eingabezeile ::=  <|datenelement|>
                      |_          _|  ,

datenelement ::= konstante | offene zeichenreihe
```

(s. Abschn. 10.1.1.)

Die Werte der Datenelemente in dieser Eingabezeile werden der Reihe nach den Variablen in der Variablenliste zugewiesen. Indizes bei indizierten Variablen und bei string-Variablen mit Teilstringangabe werden erst dann berechnet, wenn den in der Variablenliste weiter links stehenden Variablen bereits Werte zugewiesen worden sind. Einer numerischen Variablen muss ein numerischer Wert in der Datenzeile entsprechen, einer string-Variablen eine Konstante oder eine offene Zeichenreihe. Fuer jede Variable in der Variablenliste muss ein Datenelement eingegeben werden. Koennen nicht alle Datenelemente auf einer Eingabezeile untergebracht werden, beendet man die Datenzeile mit einem Komma. Dann wird eine weitere Eingabezeile mit "? " angefordert (nur fuer die erste Zeile kann eine andere Eingabeaufforderung mit PROMPT angewiesen werden).
Die Eingabe wird auf diese Weise fortgesetzt, bis allen Variablen der Variablenliste Werte zugewiesen sind.
Ist ein Zeitstopp-Modifikator angegeben, wird die fuer die Eingabe benoetigte Zeit gemessen (in Sekunden) und der numerischen Variablen nach ELAPSED zugewiesen.
Ein Zeitlimit-Modifikator wird nicht beachtet (bei Betriebssystemen mit entsprechenden Moeglichkeiten dient der Zeitlimit-Modifikator zum Abbruch der Eingabe, wenn diese nicht in einer vorgegebenen Zeit abgeschlossen wird).
Auch bei der line-input-Anweisung wird vor der Eingabe eine Eingabeaufforderung (Wert des string-Ausdrucks nach PROMPT oder "? ") ausgegeben. Danach wird die Programmabarbeitung unterbrochen, bis eine Eingabezeile ueber das Terminal eingegeben wird. Diese gesamte Eingabezeile wird der ersten string-Variablen in der string-Variablenliste zugewiesen. Dabei gehoert das eol-Zeichen, das die Eingabezeile abschliesst, nicht zum Wert der string-Variablen. Fuehrende und nachfolgende Leerzeichen sowie Anfuehrungszeichen werden jedoch unveraendert uebernommen. Folgen zwei Anfuehrungszeichen unmittelbar aufeinander, werden diese nicht zu einem Zeichen zusammengefasst, wie das innerhalb von string-Konstanten erfolgt.

```
                                  *
line_input_eingabezeile ::= [zeichen] eol
```

Enthaelt die string-Variablenliste weitere string-Variable, wird mit "? " eine weitere line-input-Eingabezeile angefordert und nach Eingabe der string-Variablen zugewiesen. Das wird solange wiederholt, bis an jede Variable der string-Variablenliste eine Eingabezeile zugewiesen wurde.

Zeitstopp- und Zeitlimit-Modifikator werden wie bei der input-Anweisung behandelt.
In folgenden Faellen fuehrt die Eingabe mit der input- oder line-input-Anweisung zu einer trivialen Ausnahme:

- Das Datenelement, das einer numerischen Variablen zugewiesen werden soll, ist keine numerische Konstante.
- Eine Eingabezeile enthaelt weniger Datenelemente als fuer die restlichen Variablen der Variablenliste benoetigt werden und die Eingabezeile endet nicht mit einem Komma.
- Eine Eingabezeile enthaelt ueberzaehlige Datenelemente.
- Eine Eingabezeile enthaelt genau die benoetigte Zahl von Datenelementen aber endet mit einem Komma.
- Eine eingegebene numerische Konstante verursacht einen Zahlenueberlauf.
- Eine string-Konstante ist laenger als die Maximallaenge der betreffenden string-Variablen.

In all diesen Faellen wird der Fehler ueber das Terminal mitgeteilt, der bereits verarbeitete Teil der Eingabezeile wird ausgewiesen und ueber ein Menue werden verschiedene Moeglichkeiten zur Korrektur der Eingabe angeboten (siehe Anleitung fuer den Bediener). An Stelle dieser Standardbehandlung kann natuerlich auch eine eigene Fehlerbehandlung erfolgen (s. Abschn. 12.).

Beispiele

Die folgende input-Anweisung liefert fuer Eingabezeilen folgende Werte:

```
.
.
.
2010 INPUT PROMPT "Endnummer Konto, Betrag, Buchungstext":&
& Konto, Betrag, Btext□
.
.
.
```

	Eingabezeile	Werte		
		Konto	Betrag	Btext□
1	165179,75, Miete August	165179	75.00	"Miete August"
2	120113,300, "POST,FREIZUG"	120113	300.00	"POST,FREIZUG"
3	154722,1E3, HO 154238	159722	1000.00	"HO 154238"

Die fuehrenden und nachfolgenden Leerzeichen im Text der 1. Eingabezeile werden abgeschnitten. In der 2. Eingabezeile ist der Text in Anfuehrungsstriche zu setzen, da das Komma nicht Bestandteil einer offenen Zeichenreihe sein kann. Die 3. Eingabe erstreckt sich ueber zwei physische Eingabezeilen.

Das nachfolgende Programmstueck kann zur Eingabe von zweidimensionalen Feldern benutzt werden. Jedes Feldelement wird einzeln angefordert.

```
600 SUB MATINP(F(,),FELDNAME¤)
605 FOR I = LBOUND(F,1) TO UBOUND(F,1)
610   FOR J = LBOUND(F,2) TO UBOUND(F,2)
615     INPUT PROMPT FELDNAME¤ & "("& STR¤(I) & "," & STR¤(J)&&
&       ")= " : F(I,J)
620   NEXT J
625 NEXT I
630 END SUB
```

Das folgende Programm kann als Trainingsprogramm fuer eine sichere Beherrschung der Tastatur verwendet werden. Es werden einige Zeichenreihen ausgegeben, die der Bediener schnell und fehlerfrei wieder eingeben muss. Durch Austausch der data-Anweisung kann der benutzte Zeichenvorrat veraendert werden.

```
100 PRINT "Geben sie folgende Zeichen schnell und fehlerfrei ein"
110 PRINT "Starten sie mit EOL"
115 LINE INPUT start_eol¤
120 LET fehlerzahl, zeitsumme = 0
130 DO
140   READ IF MISSING THEN EXIT DO: zeile¤
150   LINE INPUT PROMPT zeile¤ & "  ", ELAPSED zeit &
&     : quittung¤
160   LET zeitsumme = zeitsumme + zeit
170   IF zeile¤ <> quittung¤ THEN LET fehlerzahl = fehlerzahl + 1
180 LOOP
190 PRINT "Benoetigte Zeit:" ; zeitsumme ; " Sekunden, Fehler: "&
&   ; fehlerzahl
 .
 . data-Anweisungen
 .
500 END
```

10.3. Nichtformatierte Ausgabe ueber das Terminal

Fuer die Ausgabe von numerischen- und string-Daten stehen die print-Anweisung fuer die nicht formatierte Ausgabe und die print-using-Anweisung fuer die formatierte Ausgabe zur Verfuegung. Die Laenge der Ausgabezeile sowie die Stellung der Tabulatoren bei der nichtformatierten Ausgabe kann mit der set-Anweisung eingestellt bzw. mit der ask-Anweisung abgefragt werden.

Syntax

```
print_anweisung ::= PRINT print_liste

                  ┌                                  ┐*
print_liste ::=   │ [druckelement] drucktrenner      │ [druckelement]
                  └                                  ┘
```

```
                    |_ , _|
drucktrenner ::= <|   |>
                    |_ ; _|

                    |_          _|
                    | ausdruck   |
druckelement ::= <|              |>
                    | tabulator  |
                    |_          _|
```

tabulator ::= TAB (index)

index ::= num_ausdruck

ausdruck ::= num_ausdruck | string_ausdruck

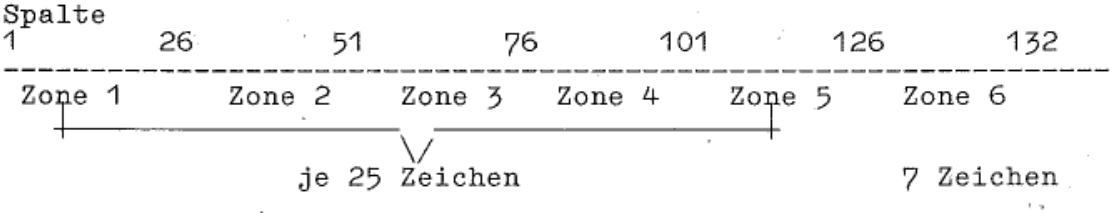
Bei der print-Anweisung werden die Druckelemente und die Drucktrenner in der print-Liste der Reihe nach von links nach rechts abgearbeitet. Der dabei entstehende Text gelangt in die Ausgabezeile. Ist das letzte Element der print-Liste kein Drucktrenner, wird abschliessend ein eol-Zeichen ausgegeben. Die Ausgabe eines eol-Zeichens in die Ausgabezeile bewirkt deren Uebertragung zum Terminal. Ist in einer print-Anweisung keine print-Liste angegeben, so wird nur ein eol-Zeichen ausgegeben.
In den Ausdruecken der print-Liste duerfen keine Funktionen aufgerufen werden, die selbst Werte mit der print-Anweisung ausgeben oder die Terminalausgaben durch trace-Anweisungen (s. Abschn. 12.) verursachen, da es sonst zu einer Vermischung von Ausgabezeilen kommt.

10.3.1. Aufbau der Ausgabezeile

Die Ausgabezeile fuer nicht formatierte Terminalausgaben hat eine bestimmte Laenge, die im folgenden als Zeilenbreite bezeichnet wird. Diese Zeilenbreite kann durch die set-margin-Anweisung festgelegt werden (s. Abschn. 10.3.2.). Standardmaessig betraegt sie 132 Zeichen.
Die Ausgabezeile ist in Druckzonen unterteilt. Die Druckzonen haben eine einheitliche Laenge, lediglich die letzte Druckzone kann kuerzer sein, wenn die Zeilenbreite kein ganzzahliges Vielfaches der Zonenlaenge ist. Die Zonenlaenge kann durch die set-zonewidth-Anweisung eingestellt werden (s. Abschn. 10.3.2.). Standardmaessig betraegt sie 25 Zeichen.
Die einzelnen Positionen in der Ausgabezeile werden als Spalten bezeichnet und von 1 bis zur Zeilenbreite durchnumeriert.

Beispiel

Eine Ausgabezeile hat standardmaessig folgenden Aufbau:



```
Spalte
1        26        51        76        101       126       132
-------------------------------------------------------------------
 Zone 1      Zone 2     Zone 3     Zone 4     Zone 5     Zone 6
   |-------------------------------------------------------|
                         \/
                  je 25 Zeichen                          7 Zeichen
```

Die Belegung der Ausgabezeile wird durch einen Zeiger gesteuert, der bei Programmstart auf Spalte 1 steht und der bei Ausgaben entsprechend weitergestellt wird. Die durch den Zeiger bezeichnete Position in der Ausgabezeile ist die aktuelle Ausgabeposition.
Nach Ausgabe eines eol-Zeichens erfolgt die Uebertragung der Ausgabezeile zum Terminal. Danach ist die aktuelle Ausgabeposition gleich 1.

10.3.2. Veraenderung und Abfrage der Zeilenbreite und der Zonenlaenge

Syntax

```
set_anweisung ::= SET set_objekt

                  ┌          ┐
                  │ MARGIN   │
set_objekt ::= <  │          │> index
                  │ ZONEWIDTH │
                  └          ┘

index ::= num_ausdruck

ask_anweisung ::= ASK ask_liste

                  ┌ ┌          ┐              ┐ *
                  │ │ MARGIN   │              │
ask_liste ::= <   │ <│          │> num_variable │>
                  │ │ ZONEWIDTH │              │
                  └ └          ┘              ┘ ,
```

Durch eine set_Anweisung wird zunaechst der Wert des Index berechnet und auf den naechsten ganzzahligen Wert gerundet. Ist als set-Objekt MARGIN angegeben, wird dieser Wert zur neuen Zeilenbreite, ist als set-Objekt ZONEWIDTH angegeben, wird der Wert zur neuen Zeilenlaenge. Mit der ask-Anweisung koennen die aktuellen Werte der Zonenlaenge und der Zeilenbreite abgefragt werden.
Sie werden der numerischen Variablen nach dem Schluesselwort MARGIN bzw. ZONEWIDTH zugewiesen.
Bei der Abarbeitung einer set-Anweisung entsteht eine nicht triviale Ausnahme wenn:

- die Zeilenbreite auf einen Wert kleiner als die aktuelle Zonenlaenge oder groesser als 132 (maximale Satzlaenge) gesetzt wird,
- die Zonenlaenge auf einen Wert groesser als die Zeilenbreite oder kleiner als 1 gesetzt wird.

Beispiel

Im folgenden Beispiel wird die Zeilenbreite abgefragt und gegebenenfalls so verkleinert, dass die letzte Zone entfaellt, falls diese kleiner als die uebrigen war.

```
.
.
.
330 ASK ZONEWIDTH zonenlaenge, MARGIN zeilenbreite
340 LET zahl_volle_zonen = INT (zeilenbreite/zonenlaenge)
350 LET neue_zeilenbreite = zahl_volle_zonen * zonenlaenge
360 SET MARGIN neue_zeilenbreite
.
.
.
```

10.3.3. Abarbeitung der Druckelemente

Bei der Abarbeitung eines Druckelements (string-Ausdruck, num-Ausdruck, TAB) entsteht eine Zeichenreihe. Diese Zeichenreihe wird ab der aktuellen Ausgabeposition in die Ausgabezeile eingefuegt. Die neue aktuelle Ausgabeposition ergibt sich aus der alten Ausgabeposition plus der Laenge der erzeugten Zeichenreihe. Falls eine erzeugte Zeichenreihe nicht mehr vollstaendig in einer teilweise gefuellten Ausgabezeile untergebracht werden kann, wird diese mit einem eol-Zeichen abgeschlossen und auf dem Terminal ausgegeben. Die Zeichenreihe wird dann ab Spalte 1 der naechsten Ausgabezeile eingefuegt. Kann sie auch dann nicht vollstaendig in einer Ausgabezeile untergebracht werden, da ihre Laenge die Zeilenbreite ueberschreitet, wird sie geteilt und erscheint in zwei aufeinanderfolgenden Ausgabezeilen.
Tritt als Druckelement ein string-Ausdruck auf, wird sein Wert berechnet. Dieser Wert liefert direkt die auszugebende Zeichenreihe.
Tritt als Druckelement ein numerischer Ausdruck auf, wird sein Wert berechnet und in die Textform konvertiert. Die erzeugte Zeichenreihe beginnt mit einem Leerzeichen bei positiven Werten, mit einem Minuszeichen bei negativen Werten. Danach erscheint der Wert in ganzzahliger Darstellung, in der Darstellung als gebrochene Zahl ohne Exponent oder in der Exponentialdarstellung. Danach folgt ein abschliessendes Leerzeichen.
Welche Darstellungsart gewaehlt wird, haengt vom Wert des Ausdrucks und von der Darstellungsgenauigkeit ab (s. Abschn. 5.6.). In den folgenden Regeln beziehen sich die Zahlenangaben ausserhalb der Klammern auf die einfache numerische Genauigkeit (OPTION ARITHMETIC NATIVE) und die Zahlen in den Klammern auf die doppelte Genauigkeit (OPTION ARITHMETIC DECIMAL).

- Ganzzahlige Werte erscheinen in der Darstellung als ganze Zahl, wenn sie mit hoechstens 6 (16) Ziffern dargestellt werden koennen. Sonst erscheinen sie in Exponentialdarstellung.
  Fuer die Darstellung als ganze Zahl sind mit Vorzeichen und abschliessendem Leerzeichen maximal 8 (18) Stellen erforderlich.

- Gebrochene Werte erscheinen in der Darstellung als gebrochene Zahl ohne Exponent, falls sie mit hoechstens 6 (16) Ziffern ohne Genauigkeitsverlust dargestellt werden koennen. Sonst erscheinen sie in Exponentialdarstellung. Die Darstellung als gebrochene Zahl ohne Exponent fordert mit Vorzeichen, Dezimalpunkt und abschliessendem Leerzeichen maximal 9 (19) Stellen.

- Bei der Darstellung in Exponentialform erscheint nach dem Leerzeichen/Minuszeichen die Mantisse mit maximal 6 (16) Stellen (Dezimalpunkt stets nach der ersten Stelle), dann der Buchstabe

E, das Vorzeichen des Exponenten und der Exponent mit 2 (3) Stellen. Diese Darstellung fordert maximal 13 (24) Stellen.

Beispiel

Fuer das folgende Programm werden Eingaben und Ausgaben in der nachfolgenden Uebersicht zusammengestellt.

```
10 PROGRAM nichtformatiertes_print
15 OPTION ARITHMETIC NATIVE
20 LET Zahl = 1
30 DO WHILE zahl < > 0
40   INPUT PROMPT "Geben sie eine Zahl ein: ":Zahl
50   PRINT "***"; zahl; "***"; zahl * 1000 &
&          "***"; zahl / 1000; "***"
60 LOOP
70 END
```

Eingabe	Ausgabe
1.42	*** 1.42 *** 1420 *** 0.0142 ***
-0.142	***-.142 ***-142 *** -.000142 ***
1420	*** 1420 *** 1.42E+06*** 1.42 ***
65E7	*** 6.5E+8 *** 6.5E+11 *** 650000 ***

Tritt als Druckelement in der print-Liste ein Tabulator auf, wird der Wert des Ausdrucks nach TAB berechnet und ggf. auf die naechste Zahl gerundet. Diese Zahl bezeichnet die zu erreichende Spalte in der Ausgabezeile. Dann werden soviele Leerzeichen ausgegeben, bis die aktuelle Ausgabeposition die geforderte Spalte erreicht. Stimmt die aktuelle Ausgabeposition bereits vorher mit der geforderten Spalte ueberein, wird kein Leerzeichen erzeugt. Ist die aktuelle Ausgabeposition vor der Abarbeitung des Tabulators groesser als die zu erreichende Spalte, wird ein eol-Zeichen in die Ausgabezeile eingefuegt, die Zeile wird ausgegeben und die geforderte Ausgabeposition wird in der neuen Zeile eingestellt. Bezeichnet der Wert des Ausdrucks nach TAB keine zwischen 1 und der Zeilenbreite liegende Spalte, so wird wie folgt verfahren:

- Ist der Wert groesser als die Zeilenbreite, wird von ihm solange die Zeilenbreite subtrahiert, bis sich ein Wert im Intervall (1, Zeilenbreite) ergibt.
- Ist der Wert kleiner als 1, entsteht eine triviale Ausnahme. Es wird der Wert 1 angenommen und fortgesetzt.

Beispiel

Das folgende Beispiel gibt die Tabelle der Zweierpotenzen aus.

```
10  Program zweierpotenz
20  LET horizontale_linie$= REPEAT$("-",39)
30  PRINT TAB(9); "Tabelle der Zweierpotenzen"
40  PRINT horizontale_linie$
50  PRINT "|    i";TAB(7); "|  2^i"; TAB(17); "|       2^-i" &
&           ; TAB(39); "|"
60  PRINT horizontale_linie$
70  LET i1, i2 = 1
80  FOR i = 1 TO 16
90    LET i1 = i1 * 2
100   LET i2 = i2 / 2
110   PRINT "| "; i; TAB(7); "| "; i1;TAB(17); "| "; i2 &
&               ;TAB(39); "|"
120 NEXT i
130 PRINT horizontale_linie$
140 END
```

Ergebnis:

Tabelle der Zweierpotenzen

i	2^i	2^-i
1	2	.5
2	4	.25
3	8	.125
4	16	.0625
5	32	.03125
6	64	.015625
7	128	.0078125
8	256	.00390625
9	512	.001953125
10	1024	.0009765625
11	4096	.000244140625
13	8192	.0001220703125
14	16384	.00006103515625
15	32768	.000030517578125
16	65536	.0000152587890625

10.3.4. Abarbeitung der Drucktrenner

Wird als Drucktrennzeichen zwischen zwei Druckelementen ein Semikolon verwendet, so werden die durch die beiden Druckelemente erzeugten Zeichenreihen unmittelbar aufeinanderfolgend in die Ausgabezeile uebernommen.
Steht ein Semikolon am Ende der print-Liste, wird dadurch die Ausgabe des eol-Zeichens unterdrueckt, das normalerweise nach dem letzten Druckelement ausgegeben wird. Die Ausgabezeile bleibt dann unvollstaendig und erscheint solange nicht auf dem Terminal, bis sie durch weitere print-Anweisungen komplettiert wird oder bis das Programm endet.

Beispiel

print-Anweisungen(en)	Druckbild
PRINT "***";"+++"	******+++
PRINT "***";1	*** 1
PRINT "***";-1	***-1
PRINT "***" PRINT "+++"	*** +++
PRINT "***"; PRINT "+++"	******+++

Wird als Drucktrennzeichen zwischen zwei Druckelementen ein Komma verwendet, so haengt die Abarbeitung davon ab, ob die aktuelle Ausgabeposition vor der letzten Druckzone liegt, oder nicht. Liegt die Ausgabeposition nach Ausgabe der Zeichenreihe des vorangehenden Druckelements vor der letzten Druckzone, erzeugt das Drucktrennzeichen Komma soviele Leerzeichen, bis der Beginn der naechsten Druckzone erreicht wird. Liegt die aktuelle Druckposition in oder nach der letzten Zone (bei vollstaendig gefuellter Ausgabezeile), so wird ein eol-Zeichen ausgegeben und die Zeile zum Terminal uebertragen.
Endet die print-Liste mit einem Komma, so erfolgt wie beschrieben der Uebergang zu einer neuen Druckzone oder Zeile und es wird die Ausgabe des eol-Zeichens am Ende der print-Anweisung unterdrueckt.

Beispiel

print-Anweisung	Ausgabe (ZONEWIDTH = 10) Spalte:		
	1	11	21
PRINT "***","+++"	***	+++	
PRINT "***", PRINT "+++"	***	+++	
PRINT TAB(3),"***",+++"		***	+++

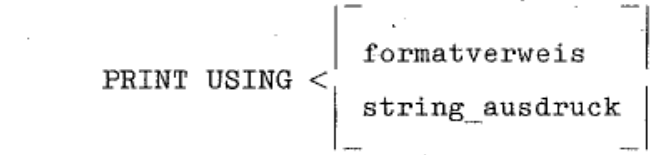
10.4. Formatierte Ausgabe ueber das Terminal

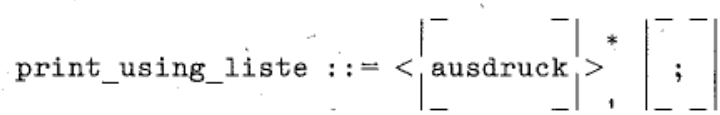
Bei der formatierten Ausgabe ueber das Terminal wird die Art der Darstellung von numerischen- und string-Werten durch Formate beschrieben. Dadurch ist eine anspruchsvollere Durckbildgestaltung als bei der nichtformatierten Ausgabe moeglich. Die Formate werden durch Folgen von Formatzeichen in einem Formatstring gebildet. Der Formatstring kann entweder direkt in der PRINT-Anweisung enthalten sein, oder er wird separat in einer image-Zeile definiert.

Syntax

---

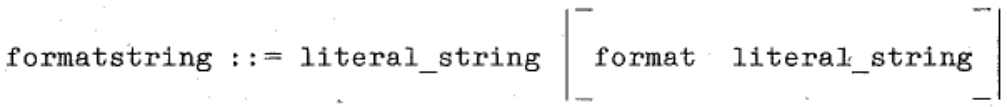
print_anweisung ::=

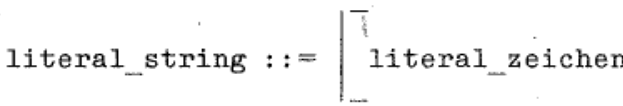


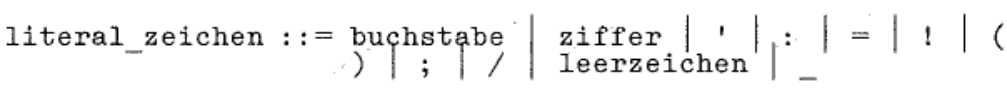


format_verweis ::= zeilennummer

image_zeile ::= zeilennummer IMAGE: formatstring







10.4.1. Abarbeitung der print-using-Anweisung

Bei der print-using-Anweisung werden die Werte der Ausdruecke der print-using-Liste mit den zugehoerigen Formaten aus dem Formatstring aufbereitet. Die dabei entstehenden Zeichenreihen werden in die Ausgabezeile eingefuegt. Die Zuordnung der Formate zu den Ausdruecken erfolgt der Reihe nach, d.h. mit dem ersten Format wird der erste Ausdruckswert aufbereitet usw. Sind bereits alle Formate zur Aufbereitung von Werten verwendet worden und es folgen weitere Ausdruecke, wird die aktuelle Ausgabezeile durch ein eol-Zeichen abgeschlossen und ausgegeben. Die Werte der uebrigen Ausdruecke werden aufbereitet, indem die Formatstring wieder von Anfang an verwendet wird. Die in der Formatstring stehenden Literalstrings werden an den entsprechenden Stellen unveraendert in die Ausgabezeile uebernommen. Enthaelt ein Formatstring mehr Formate, als zur Aufbereitung der verbleibenden Ausdruckswerte benoetigt werden, so wird ein evtl. nach dem letzten benutzten Format vorkommender Literalstring noch in die Ausgabezeile uebernommen. Anschliessend wird die Ausgabe abgeschlossen.
Bei Abschluss der Ausgabe wird ein eol-Zeichen in die Ausgabezeile eingefuegt und die Zeile wird zum Terminal uebertragen, falls die formatierte print-Liste nicht mit einem Semikolon endet. In diesem Fall bleibt die Ausgabezeile unvollendet.
Durch Benutzung des Semikolons am Ende der print-Liste bzw. der formatierten print-Liste kann ein und die selbe Ausgabezeile schrittweise durch print- und print-using-Anweisungen aufgebaut

werden. Die automatische Einfuegung eines eol-Zeichens bei Ueberschreitung der Zeilenbreite erfolgt dann nur in der print-Anweisung, nicht in der print-using-Anweisung. Fuer die print-using-Anweisung hat die Beschraenkung der Zeilenbreite keine Wirkung.
Die maximale Satzlaenge (s. Abschn. 11.) darf jedoch nicht ueberschritten werden.
Enthaelt ein Formatstring kein Format, so darf keine print-using-Liste angegeben werden, sonst wird eine nicht triviale Ausnahme verursacht.
Die Formatstring kann nach USING entweder direkt angegeben werden (als Stringausdruck) oder besser als Formatverweis. Ist der Formatstring durch einen Formatverweis gegeben, muss dieser Formatverweis aus der Zeilennummer einer image-Zeile in der selben Programmeinheit bestehen. Die Formatstring in der image-Zeile beginnt mit dem Zeichen nach dem Doppelpunkt und endet mit dem Zeichen vor dem eol-Zeichen. Die Formate in einer image-Zeile werden bereits durch den Uebersetzer analysiert, so dass die Ausgabe in diesem Fall wesentlich schneller erfolgt, als bei direkter Angabe des Formatstrings in der print-using-Anweisung, bei der die Formate zur Laufzeit ausgewertet werden muessen.
Wird bei der Abarbeitung eine image-Zeile erreicht, wird sie ohne Wirkung uebergangen.

Beispiele

In den folgenden Beispielen wird als Format die Zeichenfolge **** verwendet, mit der ein numerischer Wert < 10000 oder eine Zeichenkette bis zu vier Zeichen aufbereitet werden kann.

- gleiche Anzahl Formate und Ausdruecke

```
PRINT USING "x-Wert ****  y-Wert ****":1,2
```

liefert

```
x-Wert    1  y-Wert    2
```

- mehr Ausdruecke als Formate

```
PRINT USING "x-Wert ****  y-Wert ****":1,2,16,3412
```

liefert

```
x-Wert    1  y-Wert    2
x-Wert   16  y-Wert 3412
```

- mehr Formate als Ausdruecke (Bei wiederholter Formatbenutzung)

```
PRINT USING "x-Wert ****  y-Wert ****":5,7,9999
```

liefert

```
x-Wert    5  y-Wert    7
x-Wert 9999  y-Wert
```

- kein Format

```
PRINT USING "-------"
```

liefert

```
-------
```

- Abschluss mit Semikolon

```
PRINT USING "x-Wert ****":1;
PRINT "  y-Wert";2
```

liefert

```
x-Wert    1  y-Wert 2
```

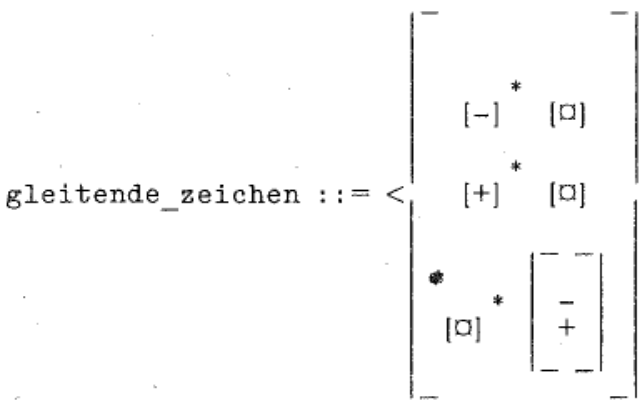
10.4.2. Aufbereitung numerischer Werte

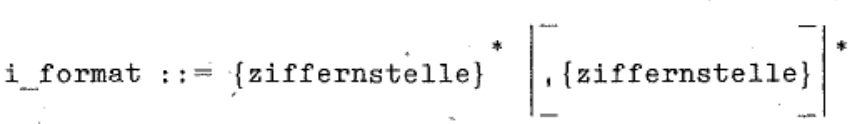
Fuer die Aufbereitung von numerischen Werten stehen drei verschiedene Formen zur Auswahl:

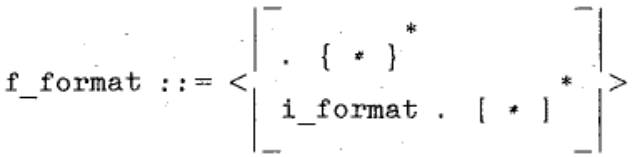
- Ausgabe des Wertes als ganze Zahl mit dem i-Format,
- Ausgabe des Wertes in der Darstellung als gebrochene Zahl ohne Exponent mit dem f-Format und
- Ausgabe des Wertes in Exponentialdarstellung mit dem e-Format.

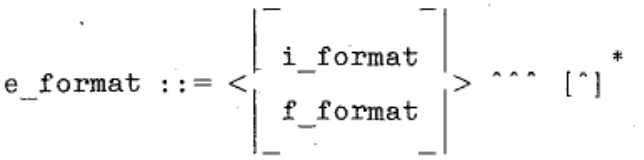
Syntax

```
format ::=
 |-                                                               -|
 |                                             |-          -|      |
 |                                             |  i_format  |      |
 |  [justagezeichen] [gleitende_zeichen]      <   f_format   >     |
 |                                             |  e_format  |      |
 |                                             |_          _|      |
<                                                                   >
 |                                                                 |
 |                                                                 |
 |  justagezeichen                                                 |
 |_                                                               _|

justagezeichen ::= < | >
```









Ein Justagezeichen hat bei der Aufbereitung numerischer Werte keine Funktion, es wird durch das nachfolgende Zeichen ersetzt. Ist das Justagezeichen das einzige Zeichen im Format oder folgt ein Punkt, so wird es durch * ersetzt.
Die gleitenden Zeichen steuern die Bildung des Vorzeichens bzw. des Waehrungszeichens vor der Zahl. Sind mehrere Zeichen +, - oder □ angegeben, wird nur eine dieser Stellen fuer das Vorzeichen oder das Waehrungszeichen benutzt, die uebrigen koennen bei Bedarf als Ziffernstelle fungieren.
Fuer die Abbildung der Ziffern der zu formatierenden Zahl stehen die Zeichen *, % und * zur Verfuegung. Je nach Wahl des Zeichens werden nichtsignifikante Nullen ausgegeben oder durch Leerzeichen oder * ersetzt. Zwischen den Ziffernstellen stehende Kommas werden unveraendert in die Ausgabezeichenreihe uebernommen, soweit links davon noch signifikante Ziffern stehen.
Die Zirkumflexzeichen ^ im e-Format besetzen Stellen fuer die Ausgabe des Exponenten. Der Dezimalpunkt in f-Format wird unveraendert in die Ausgabezeichenreihe uebernommen und bestimmt ausserdem eventuelle Rundungen bzw. Normierungen des Zahlenwertes.
Die Ausgabekette, die bei der Aufbereitung eines Wertes erzeugt wird, hat in jedem Fall die gleiche Anzahl von Zeichen wie das zur Aufbereitung verwendete Format.
Die Wirkung der einzelnen Zeichen beschreibt folgende Tabelle:

Tabelle 3: Verwendung der Formatzeichen

Zeichen	Wirkung
	Justagezeichen (werden fuer num. Werte ersetzt)
<	linksbuendige Einordnung von Zeichenreihen
>	rechtsbuendige Einordnung von Zeichenreihen (s. Abschn. 10.4.3.)
	gleitende Zeichen
+	Das Zeichen kennzeichnet eine Vorzeichenstelle. Es erscheint bei positiven Zahlen das Zeichen "+", bei negativen das Zeichen "-". Ist nur ein "+" angegeben, wird das Vorzeichen an der entsprechenden Stelle erzeugt (feste Einfuegung des Vorzeichens). Sind mehrere Zeichen "+" angegeben, wird das Vorzeichen an einer dieser Stellen ausgegeben, die uebrigen koennen als Ziffernstellen belegt werden und werden sonst mit Leerzeichen belegt.
-	Wie "+" bei positiven Werten erscheint jedoch anstelle des Vorzeichens "+" das Leerzeichen.
¤	Allein oder kombiniert mit + oder - zur festen bzw. gleitenden Einfuegung des Waehrungszeichens vor der Zahl.
,	Zur uebersichtlichen Gliederung groesserer Zahlen, z.B. in Dreiergrupen im ganzzahligen Teil der Zahl. Das Komma wird unveraendert aus dem Format uebernommen, wenn links davon noch Ziffern stehen. Sonst wird es durch Leerzeichen ersetzt.
.	Der Dezimalpunkt wird in die Ausgabezeichenreihe an gleicher Stelle wie im Format uebernommen. Aufzubereitende Werte werden entsprechend gerundet oder normiert.
	Ziffernzeichen
#	Durch dieses Zeichen wird eine Ziffernstelle reserviert. Werden zur Darstellung des ganzahligen Teils bei der Aufbereitung des Wertes weniger Ziffernstellen benoetigt als angegeben, werden anstelle der nichtsignifikanten Nullen Leerzeichen eingefuegt.
%	Wie "#", jedoch bleiben nichtsignifikante Nullen erhalten.
*	Wie "#", jedoch werden fuer nichtsignifikante Nullen Sternzeichen erzeugt.
^	Das Zirkumflexzeichen reserviert eine Exponentenstelle im e-Format. Jedes e-Format muss mindestens drei dieser Zeichen enthalten (eines fuer den Buchstaben E, eines fuer das Vorzeichen des Exponenten, eine Exponentenstelle).

Aufbereitung mit dem i-Format:

Wird zur Aufbereitung eines numerischen Wertes ein i-Format verwendet, wird der Wert zunaechst auf die naechste ganze Zahl gerundet. Dann wird die externe Darstellung der Zahl von rechts

nach links in die entsprechenden Ziffernstellen des Formats eingeordnet. Nichtsignifikante Nullen werden entsprechend des jeweiligen Ziffernzeichens behandelt (siehe Tabelle). Ist keine Vorzeichenstelle durch "+" oder "-" im Format angegeben und der auszugebende Wert ist negativ, wird das Vorzeichen in eine Ziffernstelle ausgegeben. Reichen die vorhandenen Ziffernstellen zur Ausgabe des Wertes und evtl. eines negativen Vorzeichens nicht aus, so wird eine triviale Ausnahme ausgeloest. Die Standardbehandlung dieser Ausnahme fuellt die Ausgabezeichenreihe entsprechend der Laenge des Formats mit Sternzeichen, gibt die Zeile aus und fuegt die nichtformatierte Darstellung des Wertes in die naechste Zeile ein. Die Ausgabe wird in der Spalte der neuen Zeile weitergefuehrt, an der sie bei korrektem Verlauf der Aufbereitung fortgesetzt worden waere.

Beispiele

Format	Wert	Ausgabe	
***	2.5	3	
	67.2	67	
	128.0	128	
	4790	***	Ausnahme
	-3	-3	
	-628	***	Ausnahme
%%%	26	026	
	1	001	
	0	000	
***	15	*15	
	0	**0	
	-33	-33	
-***	-628	-628	
	-1628	****	Ausnahme
	-1	- 1	
+***	75	+*75	
	0	+**0	
----*	19	19	
	366	366	
	-1289	-1289	
	10500	10500	
	-10500	*****	Ausnahme
␣␣␣␣+*	1630	␣+1630	
␣␣␣␣-*	1630	␣ 1630	
----␣*	-1630	-␣1630	
	-630	-␣630	
	630	␣630	
<****	530	530	
	-1630	-1630	

Aufbereitung mit dem f-Format:

Beim f-Format wird zunaechst ueberprueft, ob der gebrochene Teil des auszugebenden Wertes in den Ziffernstellen nach dem Dezimalpunkt untergebracht werden kann. Ist dies nicht der Fall, wird der Wert entsprechend gerundet. Der ganzzahlige Teil der Zahl wird wie beim i-Format behandelt.

Beispiele

Format	Wert	Ausgabe
.	27	27.00
	-3.456	-3.46
□□□*.**	650	□650.00
	-79.509	□-79.51

Aufbereitung im e-Format:

Beim e-Format wird die aufzubereitende Zahl so normiert, dass alle Ziffernstellen (ausser bei 0) des ganzzahligen Teils der Mantisse belegt werden. Danach wird die Mantisse entsprechend des verwendeten e- bzw. i-Formats gegebenenfalls gerundet und aufbereitet. Das erste Zirkumflexzeichen wird mit dem Buchstaben E belegt. Das zweite mit dem Vorzeichen des Exponenten (+ oder -), die folgenden mit Exponentenziffern. Fehlen Exponentenstellen zur Ausgabe signifikanter Ziffern des Exponenten, wird eine triviale Ausnahme ausgeloest (Behandlung siehe i-Format).

Beispiele

Format	Wert	Ausgabe
****.*^^^^	3.14	3410.0E-03
	0.26	2600.0E-04
	10257	1025.7E+01
	10257.7	1025.8E+01
*.***^^^^	3.14	3.140E+00
	0.26	2.600E-01
	10257	1.026E+04
	10257.7	1.026E+04
	-697	-.697E+03

10.4.3. Aufbereitung von Stringwerten

Zur Aufbereitung von Stringwerten wird ein beliebiges Format wie fuer numerische Daten verwendet. Jedes Formatzeichen des Formats stellt eine Stelle fuer ein Zeichen der auszugebenden Zeichenkette dar. Enthaelt das Format ein Justagezeichen (siehe Beispiel) wird die Zeichenkette, falls sie kuerzer als das Format ist, linksbuendig bzw. rechtsbuendig ausgerichtet.
Fehlt ein Justagezeichen und das Format ist laenger als die Zeichenreihe, wird die Zeichenreihe zentrisch ausgerichtet. Dabei wird u.U. rechts ein Leerzeichen mehr ausgegeben, wenn die Laengendifferenz zwischen Format und Zeichenreihe ungerade ist.
Ist das Format kuerzer als die auszugebende Zeichenkette, fuehrt

das zu einer nichttrivialen Ausnahme.

Beispiele

Format	Wert	Ausgabe	
<***^^^^	12345678 123456	12345678 123456	
>********	12345678 123456 ABCDEFGHI	12345678 123456 ********	Ausnahme
********	12345678 1234 12345	12345678 1234 12345	

10.4.4. Ausnahmen bei formatierter Terminalausgabe

Nicht triviale Ausnahmen:

- Ein direkt angegebenes Format (String-Ausdruck nach USING) ist syntaktisch falsch.

Triviale Ausnahmen (Standardbehandlung siehe i-Format):

- Zur Ausgabe einer ganzen Zahl bzw. des ganzzahligen Teils einer gebrochenen Zahl werden mehr Ziffernstellen benoetigt, als das Format bietet.

- Zur Ausgabe eines Exponenten werden mehr Exponentenstellen benoetigt, als Zirkumflexzeichen im e-Format fuer Exponentenziffern bereitstehen.

- Der Wert eines String-Audrucks ist laenger als das verwendete Format.

11. Dateien

Dateien sind Mengen von Daten, die sich ausserhalb eines BASIC-Programms befinden. Sie erlauben dem Nutzer, Daten waehrend eines Programmlaufes zu speichern und zu einem spaeteren Zeitpunkt waehrend eines weiteren Programmlaufes auf diese Daten zuzugreifen. Der Prozess des Transportes von Daten zwischen einem BASIC-Programm und einer Datei wird Eingabe bzw. Ausgabe genannt.
Nicht alle Ein- und Ausgaben erfolgen aus bzw. in Dateien. Die in diesem Abschnitt beschriebenen Anweisungen sind auch auf Geraete wie z.B. Terminal und Drucker anwendbar. Geraete unterscheiden sich von Dateien in zweierlei Weise. Sie muessen erstens nicht die Faehigkeit haben, Daten zwischen zwei Ausfuehrungen eines BASIC-Programms zu speichern. Zweitens muessen nicht alle Operationen, die fuer Dateien moeglich sind, auch fuer Geraete moeglich sein. Trotz der angedeuteten Differenzen, auf die noch detailliert eingegangen wird, lassen sich Ein- und Ausgaben sowie einige weitere Operationen weitgehend einheitlich fuer Dateien und Geraete beschreiben. Im folgenden werden deshalb unter dem Begriff Datei immer echte Dateien und Geraete verstanden, sofern nicht ausdruecklich der Unterschied ausgewiesen wird.
Der vorliegende Abschnitt beschreibt das logische Auftreten von Dateien und Geraeten innerhalb eines BASIC-Programms.
Es gibt zwei Arten der Dateiorganisation: sequentielle und stream-Dateien. Eine sequentielle Datei besteht aus einer Folge von Datensaetzen. Eine stream-Datei besteht aus einer Folge von Werten. Geraete koennen nur wie sequentielle Dateien verwendet werden.
Es gibt zwei Arten von Datensaetzen: Saetze in display- bzw. Textformat und Saetze im Internformat. Ein Satz im Textformat ist eine Folge von Zeichen. Ein Satz im Internformat ist eine Folge von Werten in interner Darstellung, d.h. in der Darstellung, die auch im Hauptspeicher benutzt wird. Datensaetze im Textformat enthalten Daten in einer Form, die fuer den Menschen lesbar ist. Sie sind insbesondere bei Ein- und Ausgaben ueber Geraete zu verwenden. Datensaetze im Internformat erlauben eine effektive Speicherung und Verarbeitung von Daten in echten Dateien auf externen Speichermedien wie z.B. Disketten.
Die folgende Tabelle zeigt, welche Kombinationen zwischen Dateiorganisation und Datensatzart zulaessig sind.

Tabelle 4: Dateiorganisation und Datensatzart

		Datensatzart	
		DISPLAY	INTERNAL
Dateiorganisation	SEQUENTIAL	zulaessig	zulaessig
	STREAM	verboten	zulaessig

Es ist also zu beachten, dass eine stream-Datei nur aus Werten im Internformat bestehen kann.

Im folgenden werden einige Begriffe erklaert, die in diesem Abschnitt verwendet werden.
Ein "Dateielement" ist die kleinste Einheit einer Datei, d.h. eine Datei besteht aus einer Folge von Dateielementen. Fuer sequentielle Dateien ist das Dateielement ein Datensatz, fuer stream-Dateien ein Wert.
Jede Datei besitzt einen "Dateizeiger", der waehrend der Verarbeitung der Datei stets auf ein Dateielement oder auf das Dateiende zeigt. Verweist der Dateizeiger auf den Datenanfang, so verweist er auf das erste Dateielement, falls die Datei nicht leer ist. Bei einer leeren Datei, die kein Dateielement enthaelt, sind Dateianfang und Dateiende identisch. Das Dateiende liegt unmittelbar hinter dem letzten Dateielement in der Folge der Dateielemente.
Es gibt vier Anweisungen, die eine Datei als Ganzes behandeln und deshalb "Dateioperationen" genannt werden: OPEN, CLOSE,ERASE und ASK. Es gibt fuenf Anweisungen, die einzelne Dateielemente behandeln und deshalb "Satzoperationen" genannt werden: INPUT (einschliesslich LINE INPUT), PRINT, READ, WRITE und SET mit Zeigerpositionierung. Die Satzoperationen koennen Daten in der Datei, Variable innerhalb des Programms und den Dateizeiger beeinflussen. PRINT und WRITE veraendern Daten in der Datei und den Dateizeiger. INPUT und READ veraendern Variable im Programm und den Dateizeiger. Die SET-Anweisung mit Zeigerpositionierung veraendert nur den Dateizeiger.

11.1. Dateioperationen

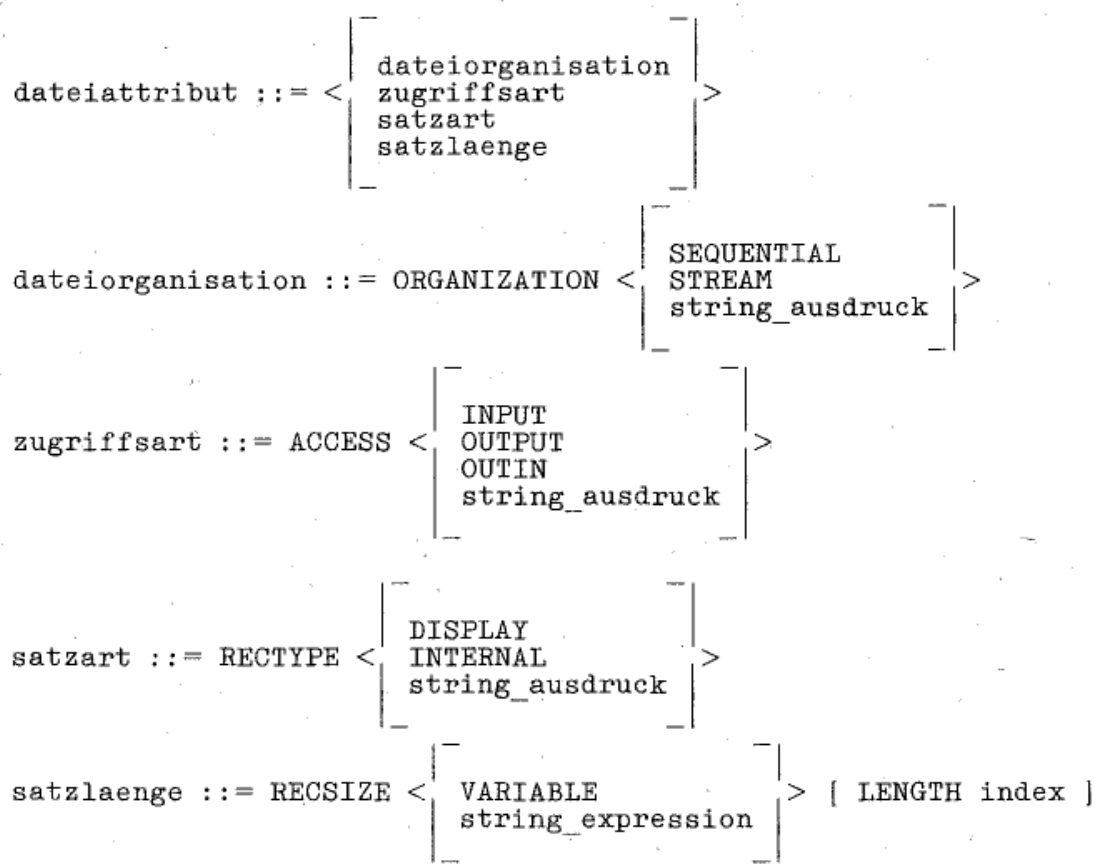
Zu den Dateioperationen zaehlen die Anweisungen OPEN,CLOSE,ERASE und ASK. Sie behandeln eine Datei als Ganzes.

11.1.1. Open-Anweisung

Die open-Anweisung ermoeglicht den Zugriff zu einer Datei, indem sie eine Verbindung zwischen der Datei und dem Programm herstellt. Jede Datei wird durch einen Dateinamen identifiziert, der gemaess den Regeln des Betriebssystems gebildet werden kann. Im Programm wird eine Datei ueber einen Kanal angesprochen, der durch eine Kanalnummer charakterisiert wird. Die open-Anweisung verbindet eine Datei gemaess eines vorgegebenen Dateinamens mit einem Kanal gemaess einer vorgegebenen Kanalnummer und ermoeglicht so den Zugriff auf die Daten der Datei.

Syntax

```
open_anweisung ::=

    OPEN kanalausdruck: NAME dateiname [,dateiattribut]

kanalausdruck ::= #index

index ::= num_ausdruck

dateiname ::= string_ausdruck
```



Der Zugriff auf Dateien erfolgt ueber Kanaele. Waehrend der Programmabarbeitung kann eine Datei einem Kanal zugeordnet (mit ihm verbunden) werden. Ein Kanal ist ein logischer Weg, ueber den Daten in ein bzw. aus einem BASIC-Programm uebertragen werden. Innerhalb einer Programmeinheit wird ein Kanal durch eine Kanalnummer identifiziert, die lokal zur Programmeinheit ist. Eine Kanalnummer muss eine ganze Zahl im Bereich von 0 bis 99 sein.
Eine Datei, die durch ihren Dateinamen identifiziert wird, heisst geoeffnet, falls sie mit einem Kanal verbunden ist. Anderenfalls heisst sie geschlossen. Ein Kanal heisst aktiv, wenn er mit einer Datei verbunden ist. Anderenfalls heisst er nicht aktiv. Bei Start eines Programms sind alle Kanaele ausser dem Kanal 0 nicht aktiv. Der Kanal 0 ist stets aktiv. Der Kanal 0 ist mit dem Bedienterminal verbunden. Das Bedienterminal hat die Dateiattribute SEQUENTIAL, DISPLAY und OUTIN und besitzt nicht die Faehigkeit, den Dateizeiger zu positionieren bzw. Daten zu loeschen. Die im Abschnitt 10. beschriebenen input- und print-Anweisungen verlaufen ueber den Kanal 0, d.h. die Angabe des Kanals 0 kann entfallen. Der Kanal 0 ist global zum gesamten Programm.
Die open-Anweisung verbindet eine Datei, die durch einen Dateinamen identifiziert wird, mit einem Kanal, der durch einen Kanalausdruck spezifiziert wird. Dateinamen werden nach den ueblichen Regeln des Betriebssystems gebildet. Die folgenden Beispiele sind gueltige Dateinamen.

```
B:TEST.DAT
ANALYSE.BAS
STAMMDAT
PRN:          (bezeichnet den Drucker)
CON:          (bezeichnet das Bedienterminal)
```

In den Dateinamen koennen wahlweise Gross- und Kleinbuchstaben verwendet werden. Die Dateinamen

TEST und Test

identifizieren also die gleiche Datei.
Die Auswertung des Kanalausdrucks muss eine zulaessige Kanalnummer (ganze Zahl im Bereich von 0 bis 99) ergeben.
Nach erfolgreicher Abarbeitung der open-Anweisung ist der spezifizierte Kanal aktiv und die Datei geoeffnet. Ist der spezifizierte Kanal bereits aktiv, so wird eine Ausnahme ausgeloest. Es koennen neben dem Kanal 0 gleichzeitig noch 4 Kanaele aktiv sein.
Durch die open-Anweisungen

```
150 OPEN #1:NAME "B:PERSONAL.DAT"
155 OPEN #2:NAME "PRN:"
```

werden die Datei PERSONAL.DAT auf dem Laufwerk B: mit dem Kanal 1 und der Drucker mit dem Kanal 2 verbunden. Die Kanaele 1 und 2 duerfen nicht aktiv sein. Nach erfolgreicher Ausfuehrung der beiden Anweisungen koennen z.B. Saetze ueber den Kanal 1 aus der Datei PERSONAL gelesen und ueber den Kanal 2, d.h. ueber Drucker ausgegeben werden.
Nach einer erfolgreichen open-Anweisung kann zu der Datei entsprechend den Dateiattributen zugegriffen werden. Die Dateiattribute werden in der open-Anweisung explizit angegeben oder es werden Standardannahmen getroffen. Durch die Dateiattribute wird festgelegt, welche Operationen mit der Datei moeglich sind. Bevor eine Datei mit dem Kanal verbunden wird, wird ueberprueft, ob die angegebenen Dateiattribute einander nicht widersprechen und ob sie zur Datei passfaehig sind. Folgende Faelle sind fehlerhaft und fuehren zu einer Ausnahme:

- Es wurde die Dateiorganisation STREAM und die Satzart DISPLAY angegeben

- Es wird versucht, ein Geraet mit der Dateiorganisation STREAM oder mit der Satzart INTERNAL zu eroeffnen.

Das erfolgreiche Eroeffnen eines Geraetes garantiert, dass mit zwei Ausnahmen durch die Datei- und Satzoperationen die gleiche Wirkung wie bei echten Dateien erzeugt wird. Bei Ausgaben werden also die gleichen Daten generiert und bei Eingaben werden die Werte bzw. Zeichen in gleicher Weise interpretiert. Bei einem Geraet kann im Gegensatz zur echten Datei der Zeiger nicht positioniert werden (s. Abschn. 11.2.) und es koennen keine Daten durch eine erase-Anweisung geloescht werden (s. Abschn. 11.1.3.).
Mit Hilfe einer ask-Anweisung (s. Abschn. 11.1.4.) kann ermittelt werden, welche Eigenschaften eine eroeffnete Datei hat.
In einer open-Anweisung darf ein Dateiattribut hoechstens einmal angegeben werden.
Wird eine Datei mehrfach, d.h. zu verschiedenen Zeitpunkten verarbeitet, so muss sie stets mit den gleichen Dateiattributen fuer die Dateiorganisation, die Satzart und die Satzlaenge, eroeffnet werden. Wird dies nicht eingehalten, so ergeben sich Fehler

waehrend der Verarbeitung der Datei.
Bei Dateien mit der Satzart INTERNAL muss ausserdem gesichert werden, dass die Verarbeitungsprogramme die gleiche Arithmetik (DECIMAL oder NATIVE) benutzen, sofern numerische Werte verarbeitet werden sollen. Ist dies nicht der Fall, so koennen numerische Daten nicht gelesen werden.
Wird der Wert eines Dateiattributes durch einen string-Ausdruck angegeben, so muss der string-Ausdruck eines der Worte ergeben, die auch direkt angegeben werden koennen. So haben z.B. die Angaben

ACCESS INPUT und ACCESS "INPUT"

die gleiche Wirkung - sie legen die Zugriffsart INPUT (Eingabe) fest. Ein string-Ausdruck nach dem Wort ACCESS muss die Werte INPUT, OUTPUT oder OUTIN liefern. Anderenfalls wird eine Ausnahme ausgeloest.
Die folgenden Ausfuehrungen beziehen sich auf die einzelnen Dateiattribute.

Dateiorganisation

Die Dateiorganisation bestimmt das logische Verhaeltnis zwischen den Dateielementen. Sie kann durch die Schluesselworte SEQUENTIAL und STREAM oder einen string-Ausdruck, der einen dieser Werte liefert, beschrieben werden. Die Dateiorganisation STREAM darf nur bei echten Dateien verwendet werden.
Eine sequentielle Datei (SEQUENTIAL) besteht aus einer Folge von Saetzen. Die Reihenfolge der Saetze wird durch die Reihenfolge bestimmt, in der die Saetze geschrieben werden. Saetze koennen nur an das Dateiende angefuegt werden. Die Identifizierung eines Satzes ueber den Dateizeiger kann nur ausgehend vom aktuellen Stand des Dateizeigers erfolgen, abgesehen davon, dass der Dateizeiger auf den Anfang (BEGIN) bzw. auf das Ende (END) der Datei eingestellt werden kann. Eine Satzoperation kann mehrere Saetze der Satzart DISPLAY, jedoch nur einen Satz der Satzart INTERNAL verarbeiten.
Eine stream-Datei (STREAM) aehnelt stark einer sequentiellen Datei. Die Dateielemente sind jedoch keine Saetze sondern einzelne Werte. Die Reihenfolge der Werte wird durch die Reihenfolge bestimmt, in der die Werte geschrieben werden. Werte koennen nur an das Dateiende angefuegt werden. Die Identifizierung eines Wertes ueber den Dateizeiger kann nur ausgehend vom aktuellen Stand des Dateizeigers erfolgen, abgesehen davon, dass der Dateizeiger auf den Anfang bzw. das Ende der Datei eingestellt werden kann. Eine Satzoperation liest oder schreibt normalerweise mehrere Werte aus bzw. in eine stream-Datei.
Ist in der open-Anweisung die Dateiorganisation nicht explizit angegeben, so wird SEQUENTIAL angenommen.
Der Kanal 0 besitzt die Dateiorganisation SEQUENTIAL.

Zugriffsart

Die Zugriffsart beschreibt die Richtung des Datentransportes zwischen der Datei und dem Programm. Sie kann durch die Schluesselworte INPUT, OUTPUT und OUTIN oder einen string-Ausdruck, der eines dieser Worte liefert, beschrieben werden.
Ist die Zugriffsart INPUT angegeben, so koennen aus der Datei Daten gelesen werden. Die Datei darf jedoch nicht veraendert

werden. Die Anweisungen INPUT, LINE INPUT, READ und SET mit Zeigerpositionierung sind erlaubt, waehrend die Anweisungen PRINT und WRITE nicht verwendet werden duerfen.
Ist die Zugriffsart OUTPUT angegeben, so koennen neue Daten in die Datei geschrieben werden. Es duerfen jedoch keine Daten gelesen werden. Die Anweisungen PRINT, WRITE und SET mit Zeigerpositionierung sind erlaubt, waehrend die Anweisungen INPUT, LINE INPUT und READ nicht verwendet werden duerfen.
Ist die Zugriffsart OUTIN angegeben, so koennen Daten aus der Datei gelesen und neue Daten in die Datei geschrieben werden. Mit Hilfe der erase-Anweisung koennen auch Teile der Datei geloescht werden. Alle Satzoperationen sind anwendbar.
Ist in der open-Anweisung die Zugriffsart nicht explizit angegeben, so wird OUTIN angenommen.
Der Kanal 0 besitzt die Zugriffsart OUTIN.
Wird eine Datei mit der Zugriffsart OUTPUT eroeffnet, so wird der Dateizeiger auf das Dateiende gesetzt. Wird sie mit INPUT oder OUTIN eroeffnet, so wird der Dateizeiger auf den Dateianfang gesetzt.

Satzart

Die Satzart beschreibt die Darstellung der Daten in den einzelnen Dateielementen. Sie legt fest, wie die Daten beim Transport zwischen Datei und Programm interpretiert und ggf. konvertiert werden. Sie kann durch die Schluesselworte DISPLAY und INTERNAL oder einen string-Ausdruck, der eines dieser Worte liefert, beschrieben werden.
Die Satzart DISPLAY besagt, dass ein Datensatz aus einer Folge von Zeichen besteht (Textsatz). Bei der Ausgabe werden die Werte in der gleichen Weise aufbereitet wie bei der print-Anweisung (s. Abschn. 10.). Bei der Eingabe werden die Zeichen in der gleichen Weise interpretiert wie bei der input-Anweisung (s. Abschn. 10.). Die Anweisungen READ und WRITE sind fuer Textsaetze moeglich, sie liefern die gleichen Ergebnisse wie die entsprechenden input- bzw. print- Anweisungen.
Die Satzart INTERNAL besagt, dass die Daten in ihrer internen Form dargestellt werden. Die interne Form eines Datums ist die Form, in der dieses Datum innerhalb des Programms (z.B. als Wert einer Variablen) gespeichert wird. Liegt eine sequentielle Datei der Satzart INTERNAL vor, so besteht jeder Datensatz aus einer Folge von Daten in interner Form. Bei einer stream-Datei wird jeder Wert (jedes Dateielement) in interner Form dargestellt. Den Daten in interner Form wird bei ihrer Abspeicherung in die Datei ein Typkennzeichen zugeordnet, so dass bei der Eingabe ueberprueft werden kann, ob der gelesene Wert an eine Variable mit zulaessigem Typ zugewiesen werden soll. Geraete koennen nicht mit der Satzart INTERNAL eroeffnet werden.
Die Satzart INTERNAL sollte bei echten Dateien vorrangig angewendet werden. Sie erlaubt eine effektive Speicherung und Verarbeitung von Daten, da bei Ein- und Ausgaben keine Konvertierung erfolgen muss.
Bei Dateien mit der Satzart INTERNAL duerfen die Anweisungen INPUT, LINE INPUT und PRINT nicht verwendet werden.
Ist in der open-Anweisung die Satzart nicht explizit angegeben, so wird DISPLAY angenommen.
Der Kanal 0 besitzt die Satzart DISPLAY.

Satzlaenge

Die Satzlaenge spezifiziert die maximale Laenge eines Datensatzes in einer Datei. Diese maximale Laenge wird nach dem Schluesselwort LENGTH angegeben. Sie muss groesser als Null sein.
Alle Dateien bestehen aus Dateielementen variabler Laenge, d.h. die Laenge der einzelnen Dateielemente ist verschieden und voneinander unabhaengig. Die Eigenschaft der variablen Satzlaenge wird durch das Schluesselwort VARIABLE oder einen string-Ausdruck, der dieses Wort liefert, beschrieben. Bei Dateien mit der Satzart DISPLAY entspricht die Satzlaenge der Anzahl der Zeichen in einem Datensatz. Bei Dateien der Satzart INTERNAL ist die Satzlaenge die Anzahl von Byte, die zur Bildung eines Dateielementes (Satz bzw. Wert) zur Verfuegung stehen. Dabei ist zu beachten, dass die einzelnen Werte entsprechend ihres Typs die folgende Anzahl von Byte zur Abspeicherung in der Datei benoetigen:

- native 5 Byte
- decimal 9 Byte
- integer 3 Byte (s. Anl.1)
- string (3 + Zeichenzahl) Byte

Wird bei einer Satzoperation versucht ein Dateielement aufzubauen und zu schreiben bzw. ein Dateielement zu lesen, dessen Laenge die maximale Satzlaenge ueberschreitet, so wird eine Ausnahme ausgeloest.
Wird in einer open-Anweisung die Satzlaenge nicht explizit angegeben, so wird angenommen, dass die Saetze variabel lang sind und die maximale Satzlaenge 132 betraegt.
Der Kanal 0 verarbeitet variabel lange Saetze von maximal 132 Zeichen Laenge.
Wuenscht der Nutzer dem Bedienterminal andere Attribute zuzuordnen, so kann er das Bedienterminal als zusaetzliche Datei eroeffnen. Zum Beispiel bewirkt die Anweisung

```
100  OPEN #1: NAME"CON:",RECSIZE VARIABLE LENGTH 80
```

die Eroeffnung des Bedienterminals und seine Verbindung mit dem Kanal 1. Das Bedienterminal kann nun sowohl ueber den Kanal 1 als auch ueber den Kanal 0 angesprochen werden. Beide Dateien haben mit Ausnahme der unterschiedlichen Satzlaengen (132 fuer Kanal 0, 80 fuer Kanal 1) die gleichen Attribute. Das Bedienterminal kann jedoch auch zusaetzlich so eroeffnet werden, dass darueber nur ein- oder nur ausgegeben werden kann.
In den folgenden Beispielen werden einige open-Anweisungen angegeben und erlaeutert, welche Eigenschaften die eroeffneten Dateien haben.

```
100  OPEN #1: NAME"B:STAMMDAT"
```

Es wird die Datei STAMMDAT auf dem Geraet B: eroeffnet und mit dem Kanal 1 verbunden. Die Datei ist sequentiell organisiert und enthaelt Saetze im Textformat mit variabler Laenge, die maximal 132 Zeichen umfassen. Es kann gelesen und geschrieben werden.

```
110  OPEN #N: NAME"PRN:", ACCESS OUTPUT, &
&              RECSIZE VARIABLE LENGTH 65
```

Es wird der Drucker als Datei eroeffnet und mit dem Kanal 5

verbunden, falls N=5 gilt. Die Datei ist sequentiell organisiert und enthaelt Textsaetze mit variabler Laenge, die maximal 65 Zeichen umfassen. Es darf nur geschrieben werden.

```
120  OPEN #3: NAME N□, ORGANIZATION SEQUENTIAL, &
&             ACCESS INPUT, RECTYPE INTERNAL
```

Es wird die Datei KOSTEN.DAT auf dem Geraet A: eroeffnet, falls N□ = "A:KOSTEN.DAT" gilt, und mit dem Kanal 3 verbunden. Die Datei ist sequentiell organisiert und enthaelt Saetze im internen Format. Die Satzlaenge ist variabel, darf aber 132 nicht ueberschreiten. Aus der Datei darf nur gelesen werden.

```
130  OPEN #8: NAME "KENNSATZ", ORGANIZATION STREAM, &
&             ACCESS OUTIN, RECTYPE INTERNAL, &
&             RECSIZE VARIABLE
```

Es wird die Datei KENNSATZ auf dem aktuellen Laufwerk eroeffnet und mit dem Kanal 8 verbunden. Die Datei ist eine stream-Datei. Die Werte werden in interner Form in der Datei dargestellt, wobei fuer die Darstellung eines Wertes maximal 132 Byte zur Verfuegung stehen. Es kann gelesen und geschrieben werden.
In den obigen Beispielen und den vorangegangenen Erlaeuterungen wurde gezeigt, wie die Dateiattribute, die nicht explizit in einer open-Anweisung aufgefuehrt werden, durch Standardannahmen ergaenzt werden. Es wird jedoch empfohlen, diese impliziten Annahmen nur in Ausnahmefaellen zu benutzen, und die Dateiattribute stets vollstaendig anzugeben. Die Lesbarkeit des Programms wird dadurch verbessert.
Die open-Anweisung ermoeglicht es, sowohl den Kanal, als auch den Dateinamen und die einzelnen Dateiattribute erst zur Laufzeit zu bestimmen. Dies wird z.B. in der folgenden open-Anweisung ausgenutzt:

```
200  OPEN #K: NAME DATEISPEZIFIKATION□, ORGANIZATION ORG□, &
&             ACCESS ZUGRIFF□, RECTYPE SATZART□, &
&             RECLENGTH VARIABLE LENGTH N+3
```

Ein solches Vorgehen kann nicht empfohlen werden, da der obigen Anweisung weder zu entnehmen ist, welcher Kanal benutzt wird, welche Datei verarbeitet wird, noch welche Attribute die Datei hat. Es sollte also versucht werden, die Angaben in der open-Anweisung moeglichst direkt zu formulieren.

Bei Abarbeitung einer open-Anweisung koennen in folgenden Faellen nicht triviale Ausnahmen auftreten:

- Der Kanalausdruck liefert einen Wert, der nicht im Bereich zwischen 0 und 99 liegt.
  (Diese Ausnahme kann auch bei allen weiteren Datei- und Satzoperationen auftreten. Sie wird deshalb im folgenden nicht mehr erwaehnt.)

- Der Kanalausdruck liefert einen Wert ungleich Null und der betreffende Kanal ist bereits aktiv.

- Ein string-Ausdruck, der ein Dateiattribut beschreiben soll, liefert keinen der vorgeschriebenen Werte.

- Die maximale Satzlaenge, die nach LENGTH angegeben wird, ist kleiner oder gleich Null.

Eine triviale Ausnahme tritt auf, wenn versucht wird, den Kanal 0 zu oeffnen.

11.1.2. Close-Anweisung

Die close-Anweisung hebt die Verbindung zwischen Datei und Programm, die durch die open-Anweisung hergestellt wurde, wieder auf.

Syntax

```
close_anweisung ::= CLOSE kanalausdruck

kanalausdruck ::= *index
```

Eine close-Anweisung loest die Verbindung zwischen Datei und Kanal. Die Datei wird geschlossen und der Kanal wird nicht aktiv. Die Anwendung einer close-Anweisung auf einen nicht aktiven Kanal fuehrt zu keiner Reaktion, d.h. die Programmabarbeitung wird ohne Fehlermeldung (Ausnahme) fortgesetzt. Beim Verlassen einer externen Funktion bzw. eines externen Unterprogramms werden alle Dateien automatisch geschlossen, die mit einem Kanal, der in der externen Funktion bzw. dem externen Unterprogramm lokal ist, verbunden. Bei Beendigung der Programmabarbeitung werden alle noch offenen Dateien automatisch geschlossen.
Das automatische Schliessen von Dateien beim Verlassen von Programmeinheiten bzw. beim Abbruch der Programmabarbeitung sichert, dass keine Daten bzw. Dateien verloren gehen. Es sollte jedoch nicht bewusst genutzt werden. Jede in einem Programm eroeffnete Datei sollte auch mit Hilfe einer close-Anweisung explizit geschlossen werden.
Durch die Anweisung

```
510  CLOSE *3
```

wird die mit dem Kanal 3 verbundene Datei geschlossen. Der Kanal 3 wird nicht aktiv.
Nach dem Schliessen einer Datei kann auf die Daten der Datei erst wieder zugegriffen werden, wenn sie erneut eroeffnet wird.

Bei der Abarbeitung einer close-Anweisung wird eine nicht triviale Ausnahme ausgeloest, falls versucht wird, die mit dem Kanal 0 verbundene Datei zu schliessen.

11.1.3. Erase-Anweisung

Mit Hilfe der erase-Anweisung koennen die Daten einer echten Datei oder ein Teil davon geloescht werden. Auf Geraete ist die erase-Anweisung nicht anwendbar.

Syntax

```
erase_anweisung ::= ERASE [REST] kanalausdruck

kanalausdruck ::= *index
```

Die erase-Anweisung loescht die Daten bzw. einen Teil der Daten in der echten Datei, die mit dem spezifizierten Kanal verbunden ist. Die Dateiattribute koennen nicht geaendert werden.
Ist das Schluesselwort REST nicht angegeben, so werden alle Daten der Datei geloescht. Die Datei wird zur leeren Datei und der Dateizeiger zeigt auf das Dateiende, welches mit dem Dateianfang uebereinstimmt.
Ist das Schluesselwort REST angegeben, so werden das Datum, auf das der Dateizeiger verweist, und alle folgenden Daten in der Datei geloescht, waehrend die vorangehenden Daten unveraendert bleiben. Der Dateizeiger zeigt dann auf das Dateiende.
Die erase-Anweisung darf nur auf Dateien angewendet werden, die mit der Zugriffsart OUTIN eroeffnet wurden. Anderenfalls wird eine Ausnahme ausgeloest.
Durch die Anweisung

```
230  ERASE #2
```

werden alle Daten in der mit Kanal 2 verbundenen Datei geloescht. Es sei ausdruecklich darauf hingewiesen, dass diese Anweisung nicht die Datei physisch beseitigt. Es wird nur der Inhalt gestrichen. Die Datei existiert als leere Datei weiter.
Durch die Anweisung

```
280  ERASE REST #5
```

werden Daten in der mit Kanal 5 verbundenen Datei gestrichen. Wieviele Daten das genau sind, haengt von der aktuellen Position des Dateizeigers ab.
Bei der Abarbeitung einer erase-Anweisung treten in folgenden Faellen nicht triviale Ausnahmen auf:

- Der spezifizierte Kanal ist nicht aktiv.

- Die mit dem spezifizierten Kanal verbundene Datei wurde nicht mit der Zugriffsart OUTIN eroeffnet.

In folgenden Faellen treten triviale Ausnahmen auf:

- Der spezifizierte Kanal ist der Kanal 0.

- Die Anweisung wird auf ein Geraet angewendet.

11.1.4. Ask-Anweisung

Mit Hilfe einer ask-Anweisung koennen die Eigenschaften einer Datei ermittelt werden.

Syntax

```
ask_anweisung ::=
     ASK kanalausdruck : { dateieigenschaft }*,
```

kanalausdruck ::= *index

dateieigenschaft ::= <
- ACCESS string_variable
- DATUM string_variable
- ERASABLE string_variable
- FILETYPE string_variable
- MARGIN num_variable
- NAME string_variable
- ORGANIZATION string_variable
- POINTER string_variable
- RECSIZE string_variable num_variable
- RECTYPE string_variable
- SETTER string_variable
- ZONEWIDTH num_ variable

>

Bei Abarbeitung einer ask-Anweisung werden die in der Anweisung aufgefuehrten Dateieigenschaften der Datei ermittelt, die mit dem spezifizierten Kanal verbunden ist. Jede Dateieigenschaft darf in einer ask-Anweisung nur einmal abgefragt werden. Die ermittelten Werte fuer die Dateieigenschaften werden in den Variablen abgespeichert, die hinter den, die Dateneigenschaft kennzeichnenden, Schluesselwoertern angegeben werden. Ist der spezifizierte Kanal nicht aktiv, d.h. mit keiner Datei verbunden, so wird allen string-Variablen ein leerer string-Wert und allen numerischen Variablen der Wert 0 zugewiesen.
In der folgenden Tabelle wird ausgewiesen, welche Werte die Variablen im einzelnen annehmen koennen und wie diese Werte zu interpretieren sind. Dabei steht A□ fuer eine string-Variable und N fuer eine numerische Variable.

Tabelle 5: Dateieigenschaften

Dateieigenschaft	Werte
ACCESS A□	beschreibt die Zugriffsart zur Datei A□ = < "INPUT" / "OUTPUT" / "OUTIN"
DATUM A□	beschreibt den Typ des Datums in der Datei, auf das der Dateizeiger zeigt A□ = < "NUMERIC" - falls ein numerisches Datum vorliegt / "STRING" - falls ein string-Datum vorliegt / "NONE" - falls kein Datum vorliegt, d.h. das Dateiende erreicht ist Die oben genannten Werte werden nur geliefert, wenn die Datei eine stream-Datei (mit interner Datendarstellung) ist. In allen anderen Faellen gilt A□ ="UNKNOWN".
ERASABLE A□	gibt an, ob Dateielemente mit der erase-Anweisung geloescht werden koennen oder nicht A□ = < "YES" / "NO"
FILETYPE A□	gibt an, ob die Datei eine echte Datei oder ein Geraet ist A□ = < "FILE" / "DEVICE"
MARGIN N	Die Variable N enthaelt die aktuelle Satzlaenge, falls die Datei eine sequentielle Datei mit Textsaetzen (ORGANIZATION SEQUENTIAL,RECTYPE DISPLAY) ist. Anderenfalls enthaelt N eine Null.
NAME A□	A□ enthaelt den Namen der Datei.
ORGANIZATION A□	beschreibt die Dateiorganisation A□ = < "SEQUENTIAL" / "STREAM"

Tabelle 5: Fortsetzung

1	2
POINTER A□	beschreibt fuer eine echte Datei die aktuelle Position des Dateizeigers A□ = < "BEGIN" – falls der Dateizeiger auf den Dateianfang zeigt "END" – falls der Dateizeiger auf das Dateiende zeigt "MIDDLE" – sonst Bei Geraeten gilt A□ = "UNKNOWN"
RECSIZE A□ N	beschreibt die Satzlaenge der Datei A□ = "VARIABLE" N enthaelt die maximale Satzlaenge
RECTYPE A□	beschreibt die Satzart der Datei A□ = < "DISPLAY" "INTERNAL"
SETTER A□	beschreibt, ob bei der vorliegenden Datei der Dateizeiger explizit positioniert werden kann A□ = < "YES" "NO"
ZONEWIDTH N	Die Variable N enthaelt die aktuelle Zonenlaenge, falls die Datei eine sequentielle Datei mit Textsaetzen (ORGANIZATION SEQUENTIAL, RECTYPE DISPLAY) ist. Anderenfalls enthaelt N eine Null.

Die Ausfuehrung einer ask-Anweisung fuer den Kanal 0 liefert folgende Werte der Dateieigenschaften:

Dateieigenschaft	Wert		
ACCESS	"OUTIN"		
DATUM	"UNKNOWN"		
ERASABLE	"NO"		
FILETYPE	"DEVICE"		
MARGIN	132		(*)
NAME	"CON:"		
ORGANIZATION	"SEQUENTIAL"		
POINTER	"UNKNOWN"		
RECSIZE	"VARIABLE"	132	(**)
RECTYPE	"DISPLAY"		
SETTER	"NO"		
ZONEWIDTH	25		(*)

(*) Dieser Wert gilt nur, falls nicht durch set-Anweisungen die aktuelle Satzlaenge bzw. die Zonenlaenge geaendert wurde.
(**) Die Festlegung von 132 Zeichen als maximale Satzlaenge fuer den Kanal 0 ist eine Abweichung von den Festlegungen des Standards. Der Standard begrenzt die maximale Satzlaenge fuer den Kanal 0 nicht.

Die ask-Anweisung kann sinnvoll verwendet werden, um Ausnahmen zu vermeiden. Im folgenden Beispiel soll eine stream-Datei verarbeitet werden, die nur string-Werte enthalten darf. Vor dem Lesen jedes Wertes wird ueberprueft, ob das folgende Datum auch wirklich ein string-Datum ist.

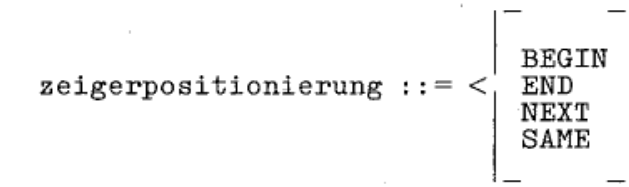
```
100  PROGRAM VERARBEITUNG
110  OPEN *1: NAME "TEXT", ORGANIZATION STREAM, ACCESS INPUT, &
&             RECTYPE INTERNAL, RECSIZE VARIABLE LENGTH 100
120  ASK *1: DATUM TYPE¤
130  DO WHILE TYPE¤ <> "NONE"
140     SELECT CASE TYPE¤
150     CASE "STRING"
160         READ X¤
170         REM  VERARBEITUNG DES GELESENEN WERTES
 .
 .
 .
300     CASE "NUMERIC"
310         PRINT "NUMERISCHES DATEIELEMENT GEFUNDEN"
320         READ X
330     END SELECT
340     ASK *1: DATUM TYPE¤
350  LOOP
360  CLOSE *1
370  END
```

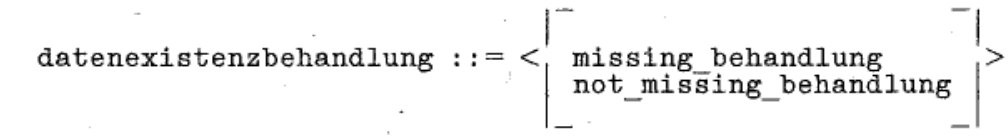
11.2. Positionierung des Dateizeigers

Der Dateizeiger einer geoeffneten Datei kann veraendert werden, ohne dass gleichzeitig ein Datentransport zwischen Programm und Datei erfolgt. Dies wird mit Hilfe der set-Anweisung mit Zeigerpositionierung realisiert. Die in diesem Abschnitt besprochenen Regeln zur Zeigerpositionierung sind auch gueltig, wenn eine Zeigerpositionierung in einer Satzoperation auftritt (s. Abschn. 11.3. und 11.4.).

Syntax

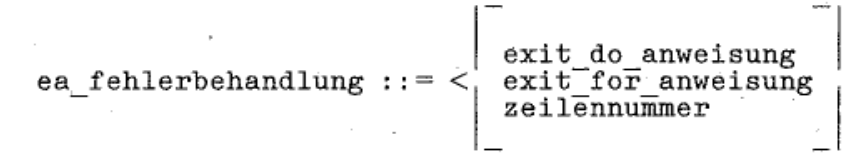
```
set_Anweisung ::= SET set_Objekt

set_Objekt ::= kanalausdruck:
   ┌                                                                  ┐
   │ POINTER zeigerpositionierung [,datenexistenzbehandlung]          │
  <│                                                                  │>
   │ datenexistenzbehandlung                                          │
   └                                                                  ┘
```





missing_behandlung ::= IF MISSING THEN ea_fehlerbehandlung

not_missing_behandlung ::= IF THERE THEN ea_fehlerbehandlung



Die Ausfuehrung einer set-Anweisung mit Zeigerpositionierung veraendert den Dateizeiger der mit dem spezifizierten Kanal verbundenen Datei. Nachdem der Dateizeiger veraendert wurde, wird die Datenexistenzbehandlung ausgefuehrt, falls eine solche angegeben wurde. Die Art und Weise der Zeigerpositionierung und der Datenexistenzbehandlung sind einheitlich fuer alle Satzoperationen (s. Abschn. 11.3. und 11.4.). Tritt waehrend der Zeigerpositionierung eine Ausnahme auf, so bleibt der Dateizeiger unveraendert. Bei Geraeten kann der Dateizeiger nicht positioniert werden. Mit Hilfe einer ask-Anweisung kann ermittelt werden, ob in einer Datei der Dateizeiger positioniert werden darf oder nicht (Dateieigenschaft SETTER).
Ist keine Zeigerpositionierung angegeben, so bleibt der Dateizeiger unveraendert. Es wird nur die Datenexistenzbehandlung ausgefuehrt.
Die Zeigerpositionierung NEXT setzt den Dateizeiger auf das naechste zu verarbeitende Dateielement (falls ein solches existiert) bzw. das Dateiende. Bei stream-Dateien zeigt der Dateizeiger stets auf dieses Dateielement, so dass er durch NEXT nicht veraendert wird. Das gleiche trifft fuer sequentielle Dateien zu. Hier gibt es jedoch eine Ausnahme: Wurde durch eine print-Anweisung ein Teilsatz erzeugt (s. Abschn. 11.3.) und enthaelt die naechste Satzoperation fuer diese Datei die Zeigerpositionierung NEXT, so wird ein Satzendezeichen (EOR) generiert, der Satz ausgegeben und der Dateizeiger auf das Dateiende gesetzt.
Die Zeigerpositionierung BEGIN setzt den Dateizeiger auf den Anfang der Datei, d.h. auf das erste Dateielement. Wenn die Datei leer ist zeigt der Zeiger gleichzeitig auf das Dateiende.
Die Zeigerpositionierung END setzt den Dateizeiger auf das Ende der Datei.
Die Zeigerpositionierung SAME erlaubt den Zugriff auf das Dateielement, das in der letzten ausgefuehrten Satzoperation behandelt wurde. Voraussetzung ist jedoch, dass diese vorangehende Operation korrekt, d.h. ohne Ausnahme, abgearbeitet wurde. Ist dies nicht der Fall, so wird der Dateizeiger nicht veraendert und eine Ausnahme ausgeloest.
Kann die Zeigerpositionierung SAME ausgefuehrt werden, so wird wie folgt verfahren: Wurde vorher eine input-, read- oder set-

Anweisung mit Zeigerpositionierung ausgefuehrt, so wird der Zeiger auf das gleiche Dateielement gesetzt, auf das er auch durch diese input-, read- bzw. set-Anweisung gesetzt wurde. Wurde vorher eine print- oder write-Anweisung ausgefuehrt, so wird der Zeiger auf das erste von diesen Anweisungen erzeugte Dateielement gesetzt.
Die Zeigerpositionierung NEXT hat mit der oben erwaehnten Ausnahme bei sequentiellen Dateien keine Wirkung. Sie kann deshalb im allgemeinen weggelassen werden.
Die Zeigerpositionierung BEGIN und END sind sinnvoll bei Dateien anzuwenden, die mehrfach gelesen oder die wechselseitig gelesen und geschrieben werden sollen.
Die Zeigerpositionierung SAME erlaubt, ein Dateielement mehrfach oder ein gerade geschriebenes Dateielement zu lesen.
Im folgenden Beispiel wird ein Satz in eine sequentielle Datei ausgegeben, anschliessend gelesen und ueber Terminal ausgegeben.

```
.
.
.
370  WRITE #3: A,B,C
380  SET #3: POINTER SAME
390  READ #3: X,Y,Z
400  PRINT "AUSGEGEBENER SATZ=";X;Y;Z
.
.
.
```

Mit dem Datenzeiger ist stets eine sogenannte Datenexistenzbedingung verknuepft, die die Werte "wahr" und "falsch" annehmen kann. Immer wenn der Dateizeiger veraendert wird, wird die Datenexistenzbedingung neu ermittelt. Die Datenexistenzbedingung ist falsch, wenn der Dateizeiger auf das Dateiende zeigt. Anderenfalls ist sie wahr.
Die Datenexistenzbehandlung erfolgt nach der Zeigerpositionierung, falls eine solche angegeben wird. Ist die Datenexistenzbedingung wahr und eine not-missing-Behandlung spezifiziert, so wird die angegebene EA-Fehlerbehandlung ausgefuehrt. Ist die Datenexistenzbehandlung falsch und eine missing-Behandlung spezifiziert, so wird die angegebene EA-Fehlerbehandlung ausgefuehrt. Ansonsten fuehrt die set-Anweisung zu keinen Aktionen.
Ist die EA-Fehlerbehandlung eine exit-do- oder eine exit-for-Anweisung, so wird diese Anweisung so ausgefuehrt, wie es im Abschnitt 8. beschrieben wurde. Ist als EA-Fehlerbehandlung eine Zeilennummer angegeben, so erfolgt ein Sprung zu der entsprechenden Zeile.

Im folgenden Beispiel soll eine Datei gelesen und verarbeitet werden. Es wird gezeigt, wie die Dateiendeerkennung mit Hilfe einer set-Anweisung realisiert werden kann.

```
.
.
.
210  OPEN #4: NAME N¤, ORGANIZATION SEQUENTIAL, &
&              ACCESS INPUT, RECTYPE INTERNAL, &
&              RECSIZE VARIABLE LENGTH 45
220  DO
230     SET #4: IF MISSING THEN EXIT DO
240     READ #4: A,B,C,D,E
250
.
.
.
300  LOOP
310  CLOSE #4
.
.
.
```

Im folgenden Beispiel sei eine echte Datei mit der Zugriffsart OUTIN eroeffnet. Es soll sowohl gelesen als auch geschrieben werden. Vor Ausfuehrung einer Schreiboperation soll mit Hilfe einer set-Anweisung geprueft werden, ob der Dateizeiger auf das Dateiende zeigt, da anderenfalls die Schreiboperation zum Fehler fuehrt.

```
.
.
.
425  SET #2: IF THERE THEN 1000
430  WRITE #2: X+Y, A, Z¤
.
.
.
1000 REM FEHLERBEHANDLUNG
1010 PRINT "DATEIZEIGER NICHT AUF DATEIENDE POSITIONIERT"
.
.
.
```

Bei Ausfuehrung der set-Anweisung koennen in folgenden Faellen nicht triviale Ausnahmen auftreten:

- Der spezifizierte Kanal ist nicht aktiv.
- Die Zeigerpositionierung SAME wurde verwendet und die vorangegangene Operation mit der Datei verlief fehlerhaft.
- Die Zeigerpositionierung SAME wurde verwendet und es existiert keine vorangehende Satzoperation, d.h. die vorangehende Operation mit der Datei war das Eroeffnen (open-Anweisung).

In folgenden Faellen treten triviale Ausnahmen auf:

- Es wird eine Zeigerpositionierung fuer den Kanal 0 versucht.
- Es wird versucht, eine Zeigerpositionierung auf ein Geraet anzuwenden.

11.3. Erzeugung von Dateien

Die in diesem Abschnitt beschriebenen Anweisungen erlauben dem Nutzer im Programm erzeugte Daten in eine Datei zu schreiben. Im Falle von echten Dateien koennen diese Daten spaeter durch weitere Programme gelesen bzw. modifiziert werden. Die im Abschnitt 10. beschriebenen Moeglichkeiten zur Ausgabe von Dateien werden verallgemeinert und fuer alle Satzarten bzw. Dateiarten erklaert. Weiterhin wird die set-Anweisung mit den set-Objekten MARGIN und ZONEWIDTH behandelt, da durch diese Anweisung die Ausgabe von Daten in sequentielle Dateien mit der Satzart DISPLAY beeinflusst werden kann.

11.3.1. Genereller Ablauf einer Schreiboperation

Durch eine Schreiboperation (print- oder write-Anweisung) werden ein oder mehrere neue Dateielemente an eine Datei angefuegt. Bereits existierende Dateielemente werden nicht veraendert.

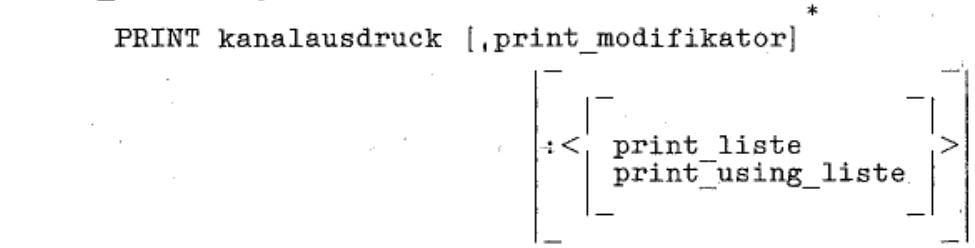
Jede Schreiboperation verlaeuft in drei Phasen.
In der ersten Phase wird der Kanalausdruck ausgewertet und der Kanal bestimmt, ueber den die Daten geschrieben werden sollen. Dann wird ueberprueft, ob die Dateiattribute die beabsichtigte Operation zulassen. Alle Schreiboperationen erfordern die Zugriffsart OUTPUT oder OUTIN. Ist der Kanal aktiv und sind die Dateiattribute zulaessig, so erfolgt ein Uebergang zur zweiten Phase. Anderenfalls wird eine Ausnahme ausgeloest und der Dateizeiger und die Datei bleiben unveraendert.
In der zweiten Phase wird der Dateizeiger eingestellt, falls eine Zeigerpositionierung angegeben ist. Dies erfolgt so, wie es im Abschnitt 11.2. beschrieben wurde. Fehlt eine solche Positionierung, so bleibt der Dateizeiger unveraendert. Anschliessend wird die Datenexistenzbedingung ermittelt (s. Abschn. 11.2.).
Ist die Datenexistenzbedingung wahr und eine not-missing-Behandlung angegeben, so wird die EA-Fehlerbehandlung (s. Abschn. 11.2.) ausgefuehrt. Ist die Datenexistenzbedingung wahr und keine not-missing-Behandlung angegeben, so wird eine Ausnahme ausgeloest. In beiden Faellen bleibt die Datei unveraendert, waehrend der Dateizeiger den Wert behaelt, der ihm durch die Zeigerpositionierung zugewiesen wurde.
Ist die Datenexistenzbedingung falsch, so beginnt die dritte Phase, die die Ausgabe der Daten gemaess der Stellung des Dateizeigers beinhaltet. Die auszugebenden Daten werden von links nach rechts berechnet bis genuegend Daten fuer das gerade zu erzeugende Dateielement vorhanden sind. Dann wird dieses Dateielement an die Datei angefuegt und der Dateizeiger hinter dieses neue Dateielement gesetzt. Tritt waehrend der Berechnung der Daten fuer das Dateielement eine Ausnahme auf, so wird das unvollstaendige Dateielement nicht an die Datei angefuegt. Wenn jedoch eine Anweisung mehrere Dateielemente erzeugt, bleiben die bereits geschriebenen Dateielemente in der Datei erhalten. Nach Abschluss der Datenausgabe zeigt der Dateizeiger auf das Dateiende.

11.3.2. Print-Anweisung

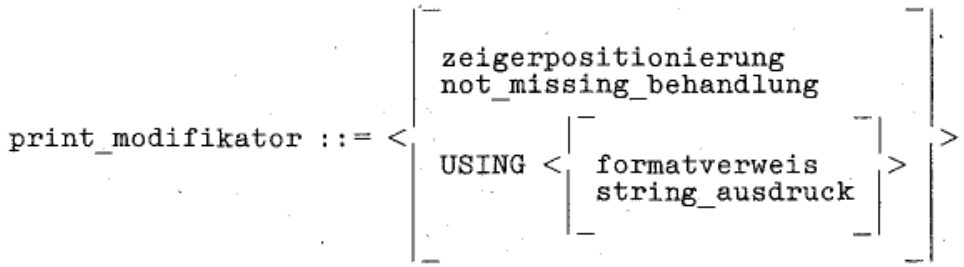
Die print-Anweisung ermoeglicht das Schreiben von Textsaetzen. Sie bietet mehr Moeglichkeiten zur Datenaufbereitung als die write-Anweisung (s. Abschn. 11.3.4.).

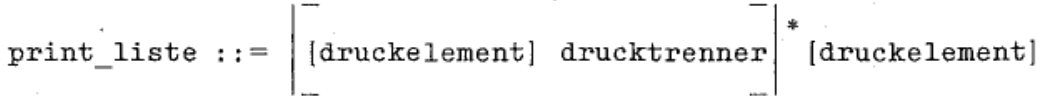
Syntax
------

print_anweisung ::=



kanalausdruck ::= *index





print_using_liste ::= {ausdruck}*, [;]

Die print-Anweisung kann nur fuer sequentielle Dateien mit der Satzart DISPLAY verwendet werden. Ihre Abarbeitung erfolgt in drei Phasen (s. Abschn. 11.3.1.) und in der dritten Phase nach den gleichen Regeln, wie sie im Abschnitt 10.3. und 10.4. beschrieben wurden. Die erzeugte Zeichenfolge bildet einen Datensatz, der mit einem eor-Zeichen (end of record) abgeschlossen wird. Die Generierung des eor-Zeichens erfolgt, wenn der Datensatz vollstaendig aufgebaut ist. Er wird dann an die Datei angefuegt. Nach diesem Anfuegen kann der Datensatz nicht mehr veraendert (z.B. verlaengert) werden. Teilsaetze werden nicht an die Datei angefuegt. Teilsaetze entstehen, wenn eine print-Liste mit einem Drucktrenner (s. Abschn. 10.3.) bzw. eine print-using-Liste mit Semikolon (s. Abschn. 10.4.) endet. Folgt eine weitere print-Anweisung ohne Zeigerpositionierung, so werden die dort erzeugten Zeichen an den vorhandenen Teilsatz angefuegt. Folgt jedoch eine print-Anweisung mit Zeigerpositionierung oder eine andere Satz- oder Dateioperation, so wird zunaechst an den Teilsatz ein eor-Zeichen angefuegt und der so erzeugte Satz ausgegeben.
Es ist moeglich mit der print-Anweisung leere Saetze, d.h. Saetze der Laenge 0 zu erzeugen.
Ein print-Modifikator darf in einer print-Anweisung nur einmal angegeben werden.
Ist als print-Modifikator eine USING-Klausel angegeben, so darf nur eine print-using-Liste verwendet werden. Fehlt diese Klausel, so darf nur eine print-Liste verwendet werden.
Die Aufbereitung der Druckelemente und die Bedeutung der Drucktrenner bei einer print-Liste sowie die Abarbeitung einer print-

using-Liste im Zusammenhang mit der zugehoerigen Formatbeschreibung erfolgt genauso, wie es in den Abschnitten 10.3. und 10.4. beschrieben wurde. Es erfolgen deshalb hier keine weiteren Erlaeuterungen.
Die Anweisung

```
170  PRINT #3, END: 10.5, 3; -18.38
```

loest folgende Aktionen aus:
Zuerst wird geprueft, ob mit dem Kanal 3 eine sequentielle Datei mit der Satzart DISPLAY verbunden ist, die Ausgaben zulaesst (Zugriffsart OUTPUT oder OUTIN). Ist dies der Fall, so wird der Dateizeiger auf das Dateiende gesetzt. Damit wird die Datenexistenzbedingung falsch und es kann ein Satz ausgegeben werden. Der Satz wird nach den in Abschnitt 10.3. beschriebenen Regeln gebildet und hat das Aussehen:

```
 10.5                          3 -18.38
```

Zeigerpositionierungen muessen nur dann angegeben werden, wenn in einer Datei wechselseitig gelesen und geschrieben wird. Solange eine Datei nacheinander durch print-Anweisungen beschrieben wird, ist eine explizite Zeigerpositionierung nicht notwendig.
Bei der Anweisung

```
280  PRINT #3, IF THERE THEN 520, USING 290: X,Y
290  IMAGE :    **.***    **.***
```

wird nach der Ueberpruefung der mit dem Kanal 3 verbundenen Datei die Datenexistenzbedingung ausgewertet. Ist sie wahr, das heisst zeigt der Dateizeiger auf einen existierenden Satz, so erfolgt eine Verzweigung zur Zeile 520. Ansonsten wird ein Satz erzeugt und ausgegeben, dessen Aufbau in der Zeile 290 beschrieben ist.
Liefert die Auswertung des Kanalausdrucks eine Null, so arbeitet die print-Anweisung genauso, als wenn kein Kanalausdruck angegeben worden waere. Zu beachten ist, dass bei der mit dem Kanal 0 verbundenen Datei der Dateizeiger nicht explizit positioniert werden darf. Die Datenexistenzbedingung fuer den Kanal 0 wird bei print-Anweisungen immer falsch gesetzt.

Bei Ausfuehrung der print-Anweisung koennen Ausnahmen auftreten. Dazu gehoeren zunaechst die Ausnahmen, die in den Abschnitten 10.3. und 10.4. beschrieben wurden. In Abhaengigkeit von den Phasen der Ausfuehrung gibt es weitere Ausnahmen.

Phase 1:

- Der angesprochene Kanal ist nicht aktiv.
- Die Datei besitzt die Satzart INTERNAL.
- Die Datei besitzt die Zugriffsart INPUT.
- Der Dateizeiger kann nicht korrekt positioniert werden (s. Abschn. 11.2.).

Phase 2:

- Die Datenexistenzbedingung ist wahr und keine not-missing-Behandlung angegeben.

Phase 3:

- Es wird versucht, einen Datensatz zu erzeugen, dessen Laenge die maximale Satzlaenge ueberschreitet. (Dies ist durch eine print-Anweisung mit USING-Angabe moeglich.)

11.3.3. Set-Anweisung

Syntax

```
set_anweisung ::= SET set_objekt

                                  |_          _|
set_objekt ::= kanalausdruck: <   | MARGIN     |> index
                                  | ZONEWIDTH  |
                                  |_          _|
```

Fuer jede Datei existiert eine maximale Satzlaenge, die entweder beim Eroeffnen explizit angegeben wird oder fuer die ein Standardwert angenommen wird (s. Abschn. 11.1.). Fuer die Ausgabe von Saetzen mit Hilfe einer print-Anweisung ohne USING-Klausel bzw. einer write-Anweisung kann wie bei der Terminalausgabe eine aktuelle Satzlaenge festgelegt werden. Die aktuelle Satzlaenge (bei der Terminalausgabe Zeilenbreite genannt, s. Abschn. 10.3.) legt fest, wie lang die Saetze bei der Ausgabe werden duerfen. Die aktuelle Satzlaenge entspricht nach der open-Anweisung der maximalen Satzlaenge. Sie kann durch eine set-Anweisung mit MARGIN-Angabe veraendert werden. Der Index nach dem Schluesselwort MARGIN gibt die neue aktuelle Satzlaenge fuer die mit dem spezifizierten Kanal verbundenen Datei an. Sie muss groesser als Null sein und darf die maximale Satzlaenge nicht ueberschreiten. Die aktuelle Satzlaenge ist gueltig, bis sie erneut durch eine set-Anweisung veraendert wird oder die Datei geschlossen wird.
Es sei ausdruecklich darauf hingewiesen, dass die aktuelle Satzlaenge nur bei der Ausgabe von Saetzen mit Hilfe der print-Anweisung ohne USING-Klausel bzw. der write-Anweisung Bedeutung hat. Bei allen anderen Satzoperationen (print-Anweisung mit USING-Klausel, input- und read-Anweisung) ist die Laenge der behandelten Saetze durch die maximale Satzlaenge begrenzt.
Die set-Anweisung mit MARGIN-Angabe darf nur auf eine sequentielle Datei mit Textsaetzen und der Zugriffsart OUTPUT oder OUTIN angewendet werden.
Fuer jede sequentielle Datei mit der Satzart DISPLAY existiert eine Zonenlaenge. Die Bedeutung der Zonenlaenge wurde in Abschnitt 10.3. beschrieben. Nach dem Eroeffnen einer Datei besitzt die Zonenlaenge den Wert 25. Sie kann durch eine set-Anweisung mit ZONEWIDTH-Angabe geaendert werden. Der Index nach dem Schluesselwort ZONEWIDTH gibt die neue Zonenlaenge fuer die mit dem spezifizierten Kanal verbundenen Datei an. Sie muss groesser als Null sein und darf die aktuelle Satzlaenge nicht ueberschreiten. Die Zonenlaenge ist gueltig, bis sie erneut durch eine set-Anweisung veraendert oder die Datei geschlossen wird.
Die set-Anweisung mit ZONEWIDTH-Angabe darf nur auf eine sequentielle Datei mit Textsaetzen und der Zugriffsart OUTPUT und OUTIN angewendet werden.
Liefert bei einer set-Anweisung die Auswertung des Kanalausdrucks den Wert Null, so arbeitet die Anweisung genauso, als wenn kein Kanalausdruck angegeben worden waere.
Wird durch eine print-Anweisung ein Teilsatz erzeugt und als

naechstes durch eine set-Anweisung die aktuelle Satzlaenge bzw. die Zonenlaenge fuer die betreffende Datei geaendert, so gelten diese geaenderten Werte bereits bei der folgenden Komplettierung des Teilsatzes. Im folgenden Beispiel

```
380  SET *4: MARGIN 100
390  SET *4: ZONEWIDTH 20
```

wird fuer die mit dem Kanal 4 verbundene Datei die aktuelle Satzlaenge von 100 Zeichen und eine Zonenlaenge von 20 Zeichen festgelegt.
Bei der Abarbeitung der set-Anweisung koennen Ausnahmen in folgenden Faellen auftreten:

- Der angesprochene Kanal ist nicht aktiv.

- Die Datei besitzt nicht die Satzart DISPLAY.

- Die Datei besitzt die Zugriffsart INPUT.

- Es wird versucht, eine aktuelle Satzlaenge einzustellen, die kleiner als die Zonenlaenge oder groesser als die maximale Satzlaenge ist.

- Es wird versucht, eine Zonenlaenge einzustellen, die kleiner als 1 oder groesser als die aktuelle Satzlaenge ist.

11.3.4. Write-Anweisung

Die write-Anweisung ermoeglicht das Schreiben von Daten fuer alle Dateiarten und alle Satzarten. Sie bietet weniger Moeglichkeiten zur Datenaufbereitung als die print-Anweisung.

Syntax

```
write_anweisung ::=
                                                  *          *
   WRITE kanalausdruck [,write_modifikator]: {ausdruck},

                        ┌                                  ┐
                        │                                  │
write_modifikator::= <  │ zeigerpositionierung             │>
                        │ not_missing_behandlung           │
                        └                                  ┘
```

Die write-Anweisung wird in drei Phasen abgearbeitet (s. Abschn. 11.3.1.). In der dritten Phase werden die Ausdruecke ausgewertet und daraus Dateielemente entsprechend der Dateiorganisation und den Satzarten aufgebaut und an die Datei angefuegt. Teilsaetze werden durch write-Anweisungen nicht erzeugt.
Ein write-Modifikator darf in einer write-Anweisung nur einmal angegeben werden.
Im folgenden wird angegeben, wieviel Dateielemente durch eine write-Anweisung erzeugt werden koennen.

Dateiorganisation SEQUENTIAL, Satzart DISPLAY:

Es koennen mehrere Saetze erzeugt werden. Hier gelten die gleichen Regeln wie bei der print-Anweisung, d.h. ein Satz wird ausgegeben, falls die aktuelle Satzlaenge durch das Hinzufuegen des naechsten Wertes ueberschritten werden wuerde. Tritt eine Ausnahme auf, so bleiben bereits an die Datei angefuegte Saetze erhalten und der Dateizeiger zeigt auf das Dateiende.

Dateiorganisation SEQUENTIAL, Satzart INTERNAL:

Jede write-Anweisung erzeugt nur einen Datensatz. Die Laenge dieses Satzes muss ausreichend sein, um die Werte aller angegebenen Ausdruecke aufzunehmen. Tritt eine Ausnahme auf, bevor die Anweisung vollstaendig abgearbeitet wurde, so wird der Satz nicht an die Datei angefuegt und der Dateizeiger hat den gleichen Wert wie nach der Phase 2.

Dateiorganisation STREAM, Satzart INTERNAL:

Eine write-Anweisung kann mehrere Dateielemente erzeugen. Fuer jeden angegebenen Ausdruck wird ein Dateielement (Wert) erzeugt. Tritt eine Ausnahme auf, so bleiben die bereits an die Datei angefuegten Dateielemente erhalten und der Dateizeiger zeigt auf das Dateiende.

Eine write-Operation fuer Saetze der Satzart DISPLAY arbeitet wie die entsprechende print-Anweisung, wobei die Kommas zwischen den Ausdruecken in der write-Anweisung die Bedeutung des Drucktrenners ";" der print-Anweisung haben. Weiterhin stehen nicht alle Moeglichkeiten, die die print-Anweisung bietet, auch in der write-Anweisung zur Verfuegung (Drucktrenner ",", USING-Klausel, Erzeugung von Teilsaetzen).
Die Anweisungen

```
100  WRITE #3: A+B, D¤, X  und
110  PRINT #3: A+B; D¤; X
```

erzeugen also identische Datensaetze.
Bei Dateien mit der Satzart INTERNAL speichert die write-Anweisung die auszugebenden Werte in ihrer internen Form und einem Typkennzeichen ab (s. Abschn. 11.1.). Es ist deshalb wie auch bei einer spaeteren Eingabe keine Konvertierung notwendig.
Die Wirkung der Anweisung

```
100  WRITE #3: A+B, D¤,X
```

haengt von den Attributen der mit dem Kanal 3 verbundenen Datei ab. Fuer eine sequentielle Datei mit Textsaetzen wurde die Wirkung bereits oben durch eine entsprechende print-Anweisung erklaert. Bei einer sequentiellen Datei mit Saetzen im Internformat wird ein Datensatz erzeugt, der zuerst einen numerischen, dann einen string- und abschliessend noch einen numerischen Wert enthaelt. Bei einer stream-Datei werden die Werte im Internformat ausgegeben.
Zu beachten ist, dass eine Datei mit numerischen Daten im Internformat zu einem spaeteren Zeitpunkt nur wieder gelesen werden kann, wenn die gleiche Arithmetik-Option wie beim Schreiben gilt. Trifft dies nicht zu, so wird eine Ausnahme ausgeloest.
Die Moeglichkeiten der Zeigerpositionierung und not-missing-Behandlung wurden bereits bei der print-Anweisung (s. Abschn. 11.3.2.) erlaeutert.
In Abhaengigkeit von der Ausfuehrungsphase koennen in folgenden

Faellen Ausnahmen auftreten.

Phase 1:

- Der angesprochene Kanal ist nicht aktiv.
- Die Datei besitzt die Zugriffsart INPUT.
- Der Dateizeiger kann nicht korrekt positioniert werden (s. Abschn. 11.2.).

Phase 2:

- Die Datenexistenzbedingung ist wahr und keine not-missing-Behandlung angegeben.

Phase 3:

- Es wird versucht, einen Datensatz zu erzeugen, dessen Laenge die maximale Satzlaenge ueberschreitet.

11.4. Eingabe aus Dateien

Die in diesem Abschnitt beschriebenen Anweisungen ermoeglichen es, Daten aus einer echten Datei, die frueher geschrieben wurde, oder von einem Geraet, zu lesen. Die im Abschnitt 10 beschriebenen Eingabemoeglichkeiten werden verallgemeinert. Weiterhin werden neue Moeglichkeiten geboten, Daten aus Saetzen aller Satzarten zu lesen.

11.4.1. Genereller Ablauf einer Eingabeoperation

Durch eine Eingabeoperation (input-, line input- bzw. read-Anweisung) werden ein oder mehrere Dateielemente aus einer Datei gelesen. Die Datei wird dabei nicht veraendert. Die Abarbeitung einer Eingabeoperation verlaeuft in drei Phasen.
In der ersten Phase wird der Kanalausdruck ausgewertet und der Kanal bestimmt, ueber den die Daten gelesen werden sollen. Dann werden die Dateiattribute der mit dem spezifizierten Kanal verbundenen Datei ueberprueft. Alle Eingabeoperationen verlangen die Zugriffsart INPUT bzw. OUTIN. Ist der Kanal aktiv und sind die Dateiattribute zulaessig, so beginnt die zweite Phase. Anderenfalls wird eine Ausnahme ausgeloest. Der Dateizeiger und die Werte der in der Eingabeoperation aufgefuehrten Variablen bleiben unveraendert.
In der zweiten Phase wird der Dateizeiger positioniert, wenn eine Zeigerpositionierung angegeben ist. Ist keine Zeigerpositionierung angegeben, so wird der Dateizeiger nicht veraendert. Die Zeigerpositionierung erfolgt, wie es im Abschnitt 11.2. beschrieben wurde. Anschliessend wird die Datenexistenzbedingung ermittelt. Ist die Datenexistenzbedingung falsch und eine missing-Behandlung angegeben, so wird die entsprechende EA-Fehlerbehandlung ausgefuehrt (s. Abschn. 11.2.). Ist die Datenexistenzbedingung falsch und keine missing-Behandlung angegeben, so wird eine Ausnahme ausgeloest. In beiden Faellen behaelt der Dateizeiger den Wert, den er nach der Zeigerpositionierung einnimmt. Die Werte der in der Eingabeoperation aufgefuehrten Variablen bleiben unveraendert.
Ist die Datenexistenzbedingung wahr, so beginnt die dritte Phase. Sie beinhaltet die Eingabe von Daten aus dem Dateielement, auf das der Dateizeiger verweist. Die Daten werden nacheinander dem

Dateielement (bzw. den Dateielementen) entnommen und den in der Eingabeoperation aufgefuehrten Variablen von links nach rechts zuordnet. Dabei ist zu beachten, dass die Indizes von indizierten Variablen und Teilstring-Angaben erst berechnet werden, nachdem an die in der Liste vorangehenden Variablen bereits Werte zugewiesen wurden und bevor ein Datum an die Variable, zu der sie gehoeren, zugewiesen wird. Es ist weiterhin zu beachten, dass die Zuweisung eines string-Wertes an eine string-Variable mit einer Teilstring-Angabe nach den Regeln erfolgt, wie sie im Abschnitt 6.5. beschrieben wurden. Tritt bei der Abarbeitung einer Eingabeoperation in der dritten Phase eine Ausnahme auf, so behalten alle Variablen, an die bereits Werte zugewiesen wurden, diese neuen Werte, waehrend alle uebrigen Variablen ihre urspruenglichen Werte behalten. Nach dem erfolgreichen Abschluss einer Eingabeoperation zeigt der Dateizeiger auf das naechste Dateielement.
Die Anzahl der von einer Eingabeoperation gelesenen Dateielemente haengt von der Art der Operation, der Dateiorganisation und der Satzart ab. Darauf wird in den folgenden Abschnitten eingegangen.
Eine SKIP REST - Angabe erlaubt bei sequentiellen Dateien, nur einen Teil der Daten eines Datensatzes zu lesen.
Waehrend der dritten Phase der Abarbeitung einer Eingabeoperation koennen Ausnahmen auftreten. Die folgende Tabelle zeigt, auf welches Dateielement der Dateizeiger nach einer solchen Ausnahme verweist.

Tabelle 6: Positionierung des Dateizeigers nach Ausnahmen bei Eingabeoperationen

Ausnahme	durch Dateizeiger referiertes Dateielement
Dateielement ist laenger als die maximale Satzlaenge	das dem zu langen Dateielement folgende Dateielement
fehlerhafte Indizes oder Teilstring-Angaben in der Variablenliste	bei sequentiellen Dateien der dem Satz, aus dem ein Wert entnommen werden muesste, folgende Satz bei stream-Dateien der Wert, der zugewiesen werden muesste
fehlerhafte Daten (unzulaessiger Typ, Syntaxfehler, Ueberlauf)	das dem Dateielement mit dem fehlerhaften Datum folgende Dateielement
zu wenig Daten in einem Dateielement	das folgende Dateielement
zu wenig Daten in der Datei	Dateiende
zu viele Daten in einem Dateielement	das folgende Dateielement

11.4.2. Input-Anweisung

Die input-Anweisung ung die line-input-Anweisung ermoeglichen die Eingabe von Textsaetzen aus sequentiellen Dateien. Sie bieten mehr Moeglichkeiten als die read-Anweisung. Sie dienen vorrangig zur Eingabe ueber Geraete.

Syntax

```
input_anweisung ::=
                                                  *
  INPUT kanalausdruck [,input_modifikator] : variablenliste
                                                    [,SKIP REST]
kanalausdruck ::= *index
                        ┌                          ┐
                        │ zeigerpositionierung     │
                        │ missing_behandlung       │
input_modifikator ::= <│ eingabeaufforderung      │>
                        │ zeitlimit_modifikator    │
                        │ zeitstopp_modifikator    │
                        └                          ┘

                                  *
variablenliste ::= {variable},

line_input_anweisung ::=

   LINE INPUT kanalausdruck [,input_modifikator]:
                                    string_variablenliste

                                                  *
string_variablenliste ::= {string_variable},
```

Jeder input-Modifikator darf in einer input- bzw. line-input-Anweisung nur einmal angegeben werden. Die Modifikatoren zur Zeigerpositionierung und zur missing-Behandlung wirken in der zweiten Phase der Abarbeitung so, wie im Abschnitt 11.4.1. beschrieben. Die Modifikatoren fuer die Eingabeanforderung, das Zeitlimit und den Zeitstopp wirken wie im Abschnitt 10.2. beschrieben, jedoch nur, wenn mit dem spezifizierten Kanal ein Terminal verbunden ist. Sie wirken also auf jeden Fall, wenn der Kanal 0 spezifiziert wird, oder wenn das Terminal zusaetzlich als Datei eroeffnet wurde (s. Abschn. 11.1.1.).
Eine input- bzw. line-input-Anweisung ist nur zulaessig, wenn die mit dem spezifizierten Kanal verbundene Datei die Dateiorganisation SEQUENTIAL und die Satzart DISPLAY besitzt. Ansonsten wird eine Ausnahme ausgeloest.
Der Datentransport erfolgt bei einer input-Anweisung genauso, wie es im Abschnitt 10.2. beschrieben wurde. An die Stelle der input-Eingabezeile tritt ein Datensatz, und das eol-Zeichen wird hier durch das eor-Zeichen ersetzt. Jedes Datum, das dem Datensatz entnommen wird, wird einer Variablen in der Variablenliste zugewiesen. Die Zuordnung erfolgt von links nach rechts.
Eine input-Anweisung kann mehrere Datensaetze verarbeiten. Ist in einem Datensatz das letzte Zeichen, das kein Leerzeichen ist, ein Komma, so wird durch die input-Anweisung ein weiterer Datensatz gelesen. Folgt auf einen solchen Datensatz das Dateiende, so wird eine Ausnahme ausgeloest. Die restlichen Variablen in der Varia-

blenliste behalten ihre alten Werte und der Dateizeiger zeigt auf das Dateiende. Es sei darauf hingewiesen, dass dieser Fall nicht durch eine missing-Behandlung behandelt werden kann. Die missing-Behandlung wird nur wirksam, wenn der Dateizeiger in der Phase 2, das heisst vor dem Lesen des ersten Datensatzes, bereits auf das Dateiende zeigt.
Ist in einer input-Anweisung die SKIP REST - Angabe angegeben, so kann der Datensatz mehr Daten enthalten, als Variablen in der Variablenliste angegeben sind. In diesem Fall werden allen Variablen Werte zugewiesen und die restlichen Daten im Datensatz ignoriert. Enthaelt ein Datensatz mehr Daten als Variable in der Variablenliste vorhanden sind und fehlt die SKIP REST - Angabe. so wird eine Ausnahme ausgeloest.
Enthaelt ein Datensatz zuwenig Daten und endet nicht mit einem Komma, so wird ebenfalls eine Ausnahme ausgeloest.
Zu beachten ist, dass bei der Eingabe von Daten, die zuvor mit einer print-Anweisung in eine echte Datei geschrieben wurden, nicht notwendigerweise die gleichen Werte ermittelt werden. Unterschiede koennen auftreten, wenn bei numerischen Daten beim Schreiben und Lesen unterschiedliche Arithmetikarten gelten oder wenn string-Werte beim Schreiben fuehrende oder nachfolgende Leerzeichen enthalten, die dann beim Lesen abgeschnitten werden.
Im folgenden Beispiel wird eine sequentielle Datei mit Textsaetzen gelesen und zur Kontrolle auf dem Drucker ausgedruckt. Jeder Satz enthaelt einen string-Wert und zwei numerische Werte.

```
.
.
.
100  OPEN #1: NAME "WERTE.DAT", ORGANIZATION SEQUENTIAL, &
&             ACCESS INPUT, RECTYPE DISPLAY, &
&             RECSIZE VARIABLE LENGTH 100
110  OPEN #2: NAME "PRN:", ORGANIZATION SEQUENTIAL, &
&             ACCESS OUTPUT, RECTYPE DISPLAY, &
&             RECSIZE VARIABLE LENGTH 100
120  DO
130    INPUT #1, IF MISSING THEN EXIT DO: A$,X,Y
140    PRINT #2: A$,X,Y
150  LOOP
160  CLOSE #1
170  CLOSE #2
.
.
.
```

Zu beachten ist, dass die Saetze der Datei WERTE.DAT zwischen ihren drei Bestandteilen (string-Wert, zwei numerische Werte) Kommas enthalten muessen, da ansonsten die input-Anweisung diese drei Bestandteile nicht erkennt und somit einen Fehler liefert. Zweckmaessiger ist an dieser Stelle eine line-input-Anweisung zu verwenden.
Eine line-input-Anweisung wird genauso abgearbeitet, wie es im Abschnitt 10.2. beschrieben wurde. An die Stelle der input-Eingabezeile tritt ein Datensatz und das eol-Zeichen wird hier durch das eor-Zeichen ersetzt. Jeder Datensatz, einschliesslich der fuehrenden und abschliessenden Leerzeichen, wird einer string-Variablen aus der string-Variablenliste als Wert zugewiesen. Die Zuordnung erfolgt von links nach rechts. Durch eine line-input-Anweisung werden soviele Datensaetze gelesen, wie Variable in der string-Variablenliste vorhanden sind. Ein leerer Datensatz fuehrt zur Zuweisung einer leeren Zeichenkette an die

betreffende Variable. Wird waehrend der Abarbeitung einer line-input-Anweisung das Dateiende erreicht und wurde noch nicht allen string-Variablen in der string-Variablenliste ein Wert zugewiesen, so behalten die restlichen Variablen ihre alten Werte, der Dateizeiger zeigt auf das Dateiende und es wird eine Ausnahme ausgeloest. Es sei darauf hingewiesen, dass dieser Fall nicht durch eine missing-Behandlung behandelt werden kann. Die missing-Behandlung wird nur wirksam, wenn der Dateizeiger in der Phase 2, d.h. vor dem Lesen des ersten Datensatzes, bereits auf das Dateiende zeigt.
Das oben behandelte Beispiel (Kontrolldruck einer Textdatei ueber Drucker) kann mit Hilfe der line-input-Anweisung wie folgt ausgedrueckt werden:

```
.
.
.
100  OPEN *1: NAME "WERTE.DAT", ORGANIZATION SEQUENTIAL, &
&             ACCESS INPUT, RECTYPE DISPLAY, &
&             RECSIZE VARIABLE LENGTH 100
110  OPEN *2: NAME "PRN:", ORGANIZATION SEQUENTIAL,
&             ACCESS OUTPUT, RECTYPE DISPLAY, &
&             RECSIZE VARIABLE LENGTH 100
120  DO
130    LINE INPUT *1, IF MISSING THEN EXIT DO: A□
140    PRINT *2: A□
150  LOOP
160  CLOSE *1
170  CLOSE *2
.
.
.
```

Ist bei einer input- bzw. line-input-Anweisung der spezifizierte Kanal der Kanal 0, so wird diese Anweisung so abgearbeitet wie die entsprechende Anweisung ohne Kanalangabe (s. Abschn. 10.2.).
Ist bei einer input- bzw. line-input-Anweisung der spezifizierte Kanal mit einem Terminal verbunden und tritt waehrend der Abarbeitung in der Phase 3 eine Ausnahme auf, so wird eine interaktive Fehlerbehandlung eingeleitet, die es dem Nutzer gestattet, den fehlerhaften Datensatz zu korrigieren. Es handelt sich um die gleiche Fehlerbehandlung, die auch bei den input-Anweisungen, wie sie im Abschnitt 10.2. beschrieben wurden, eingeleitet wird.
Ist bei einer input- bzw. line-input-Anweisung der spezifizierte Kanal mit einem Terminal verbunden, so fuehrt die Angabe einer Zeigerpositionierung zu einer trivialen Ausnahme (Geraet besitzt nicht die Faehigkeit zur Zeigerpositionierung). Die Angabe einer missing-Behandlung ist nur sinnvoll, wenn der spezifizierte Kanal nicht der Kanal 0 ist, wenn also das Terminal zusaetzlich als Datei eroeffnet wurde. In einem solchen Fall wird die Zeichenkombination Control-Z eor als Dateiende gewertet. werden. Beim Kanal 0 wird diese Zeichenkombination nicht als Dateiende akzeptiert und die Datenexistenzbedingung ist fuer input- bzw. line-input-Anweisungen stets wahr.
Waehrend der Ausfuehrung einer input- bzw. line-input-Anweisung koennen nicht triviale Ausnahmen auftreten. Sie werden im folgenden nach den Ausfuehrungsphasen geordnet aufgefuehrt. Zu beachten ist, dass bei Eingaben ueber Terminal jedoch die oben erwaehnte interaktive Fehlerbehandlung wirksam wird.

Phase 1
- Der spezifizierte Kanal ist nicht aktiv.
- Die Datei besitzt die Satzart INTERNAL.
- Die Datei besitzt die Zugriffsart OUTPUT.
- Der Dateizeiger kann nicht wie gefordert positioniert werden (s. Abschn. 11.2.).

Phase 2
- Die Datenexistenzbedingung ist falsch und eine missing-Behandlung wurde nicht angegeben.

Phase 3
- Es wird versucht, einen Satz zu lesen, der laenger als die maximale Satzlaenge ist.
- Der gelesene Datensatz enthaelt syntaktische Fehler.
- In der Variablenliste folgt eine numerische Variable, jedoch ist der naechste Wert im Datensatz keine numerische Konstante.
- Bei der Konvertierung eines numerischen Wertes entsteht ein Ueberlauf.
- Bei der Zuweisung eines gelesenen string-Wertes an eine string-Variable entsteht ein Ueberlauf.
- Ein Datensatz enthaelt nicht genuegend Daten und endet nicht mit einem Komma.
- Beim Versuch einen weiteren Datensatz zu lesen, wird das Dateiende erkannt.
- Ein Datensatz enthaelt zu viele Daten und die SKIP REST - Angabe wurde nicht angegeben.

11.4.3. Read-Anweisung

Eine read-Anweisung ermoeglicht die Eingabe von Daten aus Dateien aller Organisationsformen und Satzarten. Sie bietet fuer die Eingabe von Textsaetzen aus sequentiellen Dateien weniger Moeglichkeiten als die input-Anweisung. Sie dient vorrangig zur Eingabe aus echten Dateien.

Syntax

```
read_anweisung ::=
                                              *
  READ kanalausdruck [,read_modifikator] : variablenliste
                                              [,SKIP REST]
kanalausdruck ::= #index
                           |-                       -|
read_modifikator ::= <     |  zeigerpositionierung   |>
                           |  missing_behandlung     |
                           |_                       _|
                                       *
variablenliste ::= { variable },
```

Jeder read-Modifikator darf in einer read-Anweisung nur einmal auftreten. Die Wirkung der Modifikatoren wurde im Abschnitt 11.4.1. beschrieben.
Die read-Anweisung wird verwendet, um Daten aus Dateien mit unterschiedlichen Organisationsformen und Satzarten zu lesen. Die den Dateielementen entnommenen Daten werden nacheinander an die Variablen in der Variablenliste zugeordnet. Die Zuordnung erfolgt

von links nach rechts. Weitere Aussagen werden in Abhaengigkeit von der Organisationsform und der Satzart der Datei vorgenommen.

Organisationsform SEQUENTIAL, Satzart DISPLAY

Eine read-Anweisung arbeitet genauso wie die entsprechende input-Anweisung (s. Abschn. 11.4.2.). Allerdings koennen bei einer read-Anweisung nicht die Modifikatoren fuer die Eingabeanforderung, das Zeitlimit und den Zeitstopp verwendet werden. Im Fall von sequentiellen Dateien mit Textsaetzen sind also die Anweisungen

```
100  INPUT #1: A,B,C      und
100  READ  #1: A,B,C
```

in ihrer Wirkung identisch. Alle im Abschnitt 11.4.2. beschriebenen Regeln zur Abarbeitung einer input-Anweisung gelten auch fuer die read-Anweisung (Anzahl der verarbeiteten Saetze, Ausnahmesituationen, SKIP REST - Angabe usw.).

Organisationsform SEQUENTIAL, Satzart INTERNAL

Durch eine read-Anweisung wird genau ein Datensatz gelesen. Dieser Datensatz muss fuer alle in der Variablenliste aufgefuehrten Variablen Werte in interner Darstellung und ein Typkennzeichen enthalten. Der Typ eines Wertes muss mit dem Typ der entsprechenden Variablen uebereinstimmen. Bei numerischen Werten werden die Typen DECIMAL und NATIVE als unterschiedliche Typen behandelt. Aus diesem Grunde werden bei der Eingabe von Daten, die zuvor mit einer write-Anweisung geschrieben wurden, stets die gleichen Werte ermittelt. Stimmen die Typen nicht ueberein, so wird eine Ausnahme ausgeloest.
Enthaelt ein Datensatz mehr Daten als Variable in der Variablenliste aufgefuehrt sind, und ist die SKIP REST - Angabe angegeben, so werden die ueberfluessigen Daten ignoriert. Fehlt die SKIP REST - Angabe, so wird eine Ausnahme ausgeloest. Enthaelt ein Datensatz zuwenig Daten, so wird ebenfalls eine Ausnahme ausgeloest.

Organisationsform STREAM, Satzart INTERNAL

Durch eine read-Anweisung werden soviele Werte aus der Datei gelesen, wie Variable in der Variablenliste aufgefuehrt sind. Der Typ der Werte muss mit dem Typ der entsprechenden Variablen uebereinstimmen. Aus diesem Grunde werden bei der Eingabe von Daten, die zuvor mit einer write-Anweisung geschrieben wurden, stets die gleichen Werte ermittelt. Stimmen die Typen nicht ueberein, so wird eine Ausnahme ausgeloest.
Wird bei der Abarbeitung der read-Anweisung das Dateiende erreicht, bevor allen Variablen Werte zugewiesen wurden, so behalten die restlichen Variablen ihre alten Werte, der Dateizeiger zeigt auf das Dateiende und eine Ausnahme wird ausgeloest.
Die SKIP REST - Angabe darf fuer stream-Dateien nicht verwendet werden.

Wird bei einer read-Anweisung kein Kanalausdruck angegeben, so wird nicht wie bei allen anderen Ein- und Ausgabeanweisungen der Kanal 0 angenommen, sondern die Eingabe erfolgt aus der internen Datenliste (s. Abschn. 10.1.).

Im folgenden Beispiel wird demonstriert wie eine sequentielle Datei mit Saetzen im Internformat an eine andere Datei mit gleichen Dateiattributen angefuegt werden kann. Jeder Satz enthaelt vier numerische Werte.

```
.
.
200  OPEN #3: "KENNZ.DAT", ORGANIZATION SEQUENTIAL, &
&             ACCESS OUTPUT, RECTYPE INTERNAL, &
&             RECSIZE VARIABLE LENGTH 36
210  OPEN #5: "KORR.DAT", ORGANIZATION SEQUENTIAL, &
&             ACCESS INPUT, RECTYPE INTERNAL, &
&             RECSIZE VARIABLE LENGTH 36
220  DO
230    READ #5, IF MISSING THEN EXIT DO: A,B,C,D
240    WRITE #3 : A,B,C,D
250  LOOP
260  CLOSE #3
270  CLOSE #5
.
.
.
```

Im folgenden Beispiel seien zwei stream-Dateien gegeben, die eine aufsteigend sortierte Folge von numerischen Werten enthalten. Aus diesen zwei Dateien ist durch Mischen eine dritte Datei zu erzeugen und zur Kontrolle ueber Drucker auszugeben.

```
100  PROGRAM MISCHEN
110  OPEN #1: NAME "A:S1.DAT", ORGANIZATION STREAM, &
&             ACCESS INPUT, RECTYPE INTERNAL, &
&             RECSIZE VARIABLE LENGTH 9
120  OPEN #2: NAME "A.S2.DAT", ORGANIZATION STREAM, &
&             ACCESS INPUT, RECTYPE INTERNAL, &
&             RECSIZE VARIABLE LENGTH 9
130  OPEN #3: NAME "A.S3.DAT", ORGANIZATION STREAM, &
&             ACCESS OUTIN, RECTYPE INTERNAL, &
&             RECSIZE VARIABLE LENGTH 9
140  SET #3: POINTER END
150  REM  LESEN DER ERSTEN WERTE
160  READ #1, IF MISSING THEN 300 : S1
170  READ #2, IF MISSING THEN 350 : S2
180  REM  ZYKLUS ZUM MISCHEN DER DATEIEN
190  DO
200    IF S1 < S2 THEN
210       WRITE #3 : S1
220       READ #1, IF MISSING THEN 300 : S1
230    ELSE
240       WRITE #3 : S2
250       READ #2, IF MISSING THEN 350 : S2
260    END IF
270  LOOP
300  REM  KOPIEREN DES RESTES DER DATEI S2.DAT
310  LET N=2
320  GOTO 400
350  REM  KOPIEREN DES RESTES DER DATEI S1.DAT
360  LET N=1
400  DO
410    READ #N, IF MISSING THEN EXIT DO : S
420    WRITE #3 : S
430  LOOP
440  CLOSE #1
450  CLOSE #2
500  REM  KONTROLLDRUCK
510  OPEN #4: NAME "PRN:", ORGANIZATION SEQUENTIAL, &
&             ACCESS OUTPUT, RECTYPE DISPLAY, &
&             RECSIZE VARIABLE LENGTH 120
520  SET #3: POINTER BEGIN
530  DO
540    READ #3, IF MISSING THEN EXIT DO: S
550    PRINT #4: S
560  LOOP
570  CLOSE #3
580  CLOSE #4
590  END
```

Waehrend der Ausfuehrung einer read-Anweisung koennen nicht triviale Ausnahmen auftreten. Sie werden im folgenden nach den Ausfuehrungsphasen geordnet aufgefuehrt. Bei Eingaben ueber Terminal wird wie bei der input-Anweisung eine interaktive Fehlerbehandlung wirksam.

Phase 1
- Der spezifizierte Kanal ist nicht aktiv.
- Die Datei besitzt die Zugriffsart OUTPUT.
- Der Dateizeiger kann nicht wie gefordert positioniert werden (s. Abschn. 11.2.).
- Fuer eine stream-Datei wurde eine SKIP REST - Angabe angegeben.

Phase 2
- Die Datenexistenzbedingung ist falsch und eine missing-Behandlung wurde nicht angegeben.

Phase 3
Fuer sequentielle Dateien und Textsaetze koennen alle Ausnahmen in der Phase 3 auftreten, die auch in Abschnitt 11.4.3. fuer diese Phase angegeben wurden. Es folgen deshalb nur Ausnahmen, die bei anderer Dateiorganisation bzw. anderen Satzarten auftreten koennen.
- Es wird versucht ein Dateielement zu lesen, das laenger als die maximale Satzlaenge ist.
- Die Datei besitzt die Satzart INTERNAL und der Typ der Variablen stimmt nicht mit dem Typ des gelesenen Wertes ueberein.
- Ein Datensatz (Satzart INTERNAL) einer sequentiellen Datei enthaelt zu wenig Daten.
- Ein Datensatz (Satzart INTERNAL) einer sequentiellen Datei enthaelt zu viele Daten und die SKIP REST - Angabe wurde nicht angegeben.
- Bei einer stream-Datei wurde das Dateiende erkannt, bevor allen Variablen in der Variablenliste Werte zugewiesen wurden.

12. Ausnahmebehandlung und Testunterstuetzung

12.1. Ausnahmebehandlung

Tritt bei der Abarbeitung eine Ausnahme ein, wird die Abarbeitung des Programms unterbrochen und die Ausnahme behandelt. Normalerweise geschieht das durch die Standardbehandlung. Dabei erfolgt bei trivialen Ausnahmen eine entsprechende Meldung und ggf. eine Korrektur des Ergebnisses, wonach die Abarbeitung fortgesetzt wird. Bei nicht trivialen Ausnahmen wird der Programmlauf nach einer Fehlermitteilung abgebrochen.

Die Mittel zur Ausnahmebehandlung bieten die Moeglichkeiten, entstehende triviale und nicht triviale Ausnahmen anstelle der Standardbehandlung durch eigene Behandlungsroutinen zu bearbeiten. Dazu werden die Anweisungen, bei denen Ausnahmen behandelt werden sollen, in den when-Teil eines protect-Blockes eingeschlossen. Tritt in diesen Anweisungen eine Ausnahme ein, wird anstelle der Standardbehandlung die dem protect-Block zugehoerige Ausnahmeroutine ausgefuehrt. In der Ausnahmeroutine kann die Art der Ausnahme anhand eines Ausnahmecodes und die verursachende Anweisung anhand der Zeilennummer festgestellt werden. Die Ausnahmeroutine kann den Fehler behandeln und die Abarbeitung auf verschiedene Art und Weise fortsetzen.

Syntax

```
                    [                    ]
                    | when_use_block     |
protect_block ::= <                      >
                    | when_use_name_block |
                    [                    ]

when_use_block ::=
  when_zeile when_block use_zeile ausnahmeroutine end_when_zeile

when_use_name_block ::=
  when_use_name_zeile  when_block  end_when_zeile

when_zeile ::=
  zeilennummer WHEN EXCEPTION IN zeilenrest

                 [      ]*
when_block ::=   | block |
                 [      ]

use_zeile ::= zeilennummer USE zeilenrest

end_when_zeile ::= zeilennummer END WHEN zeilenrest

when_use_name_zeile ::=
  zeilennummer WHEN EXCEPTION USE name_ausnahmeroutine zeilenrest

name_ausnahmeroutine ::= routine_identifikator

selbstaendige_ausnahmeroutine ::=
  handler_zeile  ausnahmeroutine  end_handler_zeile
```

ausnahmeroutine ::= [block]*

handler_zeile ::=
  zeilennummer HANDLER name_ausnahmeroutine zeilenrest

end_handler_zeile ::=
  zeilennummer END HANDLER zeilenrest

Zur eigenen Behandlung von Ausnahmen stehen zwei Formen des protect-Blockes zur Verfuegung. Beim when-use-Block ist die Ausnahmeroutine direkt im protect-Block enthalten. Sie folgt nach dem Schluesselwort USE.
Beim when-use-name-Block wird in der when-use-name-Zeile der Name einer selbstaendigen Ausnahmeroutine angegeben, mit der im when-Block auftretende Ausnahmen zu behandeln sind. Ist der when-use-name Block in einer internen Prozedur enthalten, muss die selbstaendige Ausnahmeroutine in derselben internen Prozedur definiert werden. Steht der when-use-name-Block ausserhalb einer internen Prozedur muss die zugehoerige selbstaendige Ausnahmeroutine in der gleichen Programmeinheit und ebenfalls ausserhalb einer internen Prozedur definiert werden. Eine selbstaendige Ausnahmeroutine kann durch mehrere when-use-name-Zeilen referiert werden, ist also dort zu verwenden, wo Ausnahmen an verschiedene Stellen durch ein und dieselbe Behandlungsroutine behandelt werden sollen.
Zu Anweisungen die sich in einem protect-Block befinden, kann nicht von aussen verzweigt werden (mit Ausnahme der when-use- bzw. der when-use-name-Zeile). Ebenso kann nicht von aussen zu den Anweisungen einer selbstaendigen Ausnahmeroutine verzweigt werden (mit Ausnahme zu deren handler-Zeile). Aus einer Ausnahmeroutine kann nicht zu Anweisungen ausserhalb dieser Routine und nicht zur use- bzw. handler-Zeile verzweigt werden. In einer Programmeinheit darf eine selbstaendige Ausnahmeroutine eines bestimmten Namens nur einmal definiert werden.
Dieser Name darf in der Programmeinheit kein anderes Objekt bezeichnen.
In einer Ausnahmeroutine darf kein protect-Block stehen. Umgekehrt darf eine selbstaendige Ausnahmeroutine nicht in einem protect-Block deklariert werden.
Die Schachtelung eines protect-Blockes in einen when-Block ist dagegen zulaessig.
Wird bei der Programmabarbeitung eine when-use- bzw. when-use-name-Zeile erreicht, so werden die Anweisungen des when-Blocks in der entsprechenden Reihenfolge bearbeitet. Tritt im when-Block keine Ausnahme ein, wird als naechste Anweisung die Anweisung bearbeitet, die auf die zugehoerige end-when-Zeile folgt.
Tritt eine Ausnahme ein, wird zur entsprechenden Ausnahmeroutine verzweigt. Beim when-use-block ist das der nach dem Schluesselwort USE folgende Block, beim when-use-name-Block die entsprechende selbstaendige Ausnahmeroutine.
Zur Behandlung der Ausnahmen in der Ausnahmeroutine und zum Test von Ausnahmeroutinen stehen folgende Anweisungen zur Verfuegung:

Syntax
------

num_standardfunktion ::= EXLINE | EXTYPE

string_standardfunktion ::= EXTEXT$

handler_return_anweisung ::= RETRY | CONTINUE

exit_handler_anweisung ::= EXIT HANDLER

cause_anweisung ::= CAUSE EXCEPTION numeric_expression

Die numerischen Standardfunktionen EXLINE und EXTYPE sind parameterlos. EXLINE liefert die Zeilennummer, bei der die Ausnahme auftrat, EXTYPE den Ausnahmecode (s. Anl. 3.). Beide Standardfunktionen duerfen nur in Ausnahmeroutinen verwendet werden.
Die String-Standardfunktion EXTEXT$ hat als numerisches Argument einen Ausnahmecode und liefert zu diesem Ausnahemcode einen entsprechenden Fehlértext. Diese Standardfunktion kann auch ausserhalb von Ausnahmeroutinen gerufen werden.

Eine Ausnahmeroutine kann auf folgende Art beendet werden:

- Wird eine retry-Anweisung verwendet, dann wird die ausnahmeverursachende Anweisung erneut ausgefuehrt. War diese Anweisung eine Eingabeanweisung, werden bereits eingegebene Daten verworfen und neue Eingabedaten angefordert.

- Wird eine continue-Anweisung verwendet, dann wird die ausnahmeverursachende Anweisung uebergangen und die nachfolgende Anweisung abgearbeitet. Trat die Ausnahme in einer for-Zeile, einer do-Zeile, einer loop-Zeile, einer if-Zeile in einem if-Block, einer elseif-Zeile, einer select-Zeile oder einer case-Zeile auf, wird jeweils der ganze Block uebergangen.

- Erreicht die Steuerung in einer Ausnahmeroutine die end-when-Zeile bzw. die end-handler-Zeile, werden die restlichen Anweisungen des when-Blockes uebergangen. Die Abarbeitung wird mit der Anweisung nach der end-when-Zeile fortgesetzt.

- Wird zur Beendigung einer Ausnahmeroutine eine exit-handler-Anweisung verwendet, wird so verfahren, als ob der zugehoerige protect-Block nicht existierte. Sind mehrere protect-Bloecke ineinander geschachtelt, kann die Ausnahme auf diese Weise von einer "inneren" Ausnahmeroutine zur naechst aeusseren "vererbt" werden. Ist kein "aeusserer" protect-Block vorhanden, wird durch die exit-handler-Anweisung die Standardbehandlung aktiviert.

Tritt in einer Ausnahmeroutine erneut eine Ausnahme ein, wird diese in jedem Fall mit der Standardbehandlung bearbeitet.
Wenn eine nicht triviale Ausnahme in einer internen oder externen Funktion oder Subroutine auftritt und keine Ausnahmeroutine angegeben war oder eine Ausnahmeroutine mit der exit-handler-Anweisung verlassen wird, so wird diese Ausnahme an den Aufruf der Funktion "vererbt" und dort, falls moeglich, behandelt.
Dieses Vererben von Ausnahmen wird solange ausgefuehrt, bis eine Ausnahmeroutine die Ausnahme loest (es wird eine handler-return-Anweisung oder die end-handler bzw. end-when-Zeile erreicht) oder bis dabei das Hauptprogramm erreicht wird. Im Hauptprogramm wird zuletzt die Standardausnahmebehandlung ausgefuehrt.
Um unterscheiden zu koennen, ob eine Ausnahme bei einem Funktions- oder Prozedurruf (z.B. bei der Parameterberechnung) entstand, oder ob sie diesem Ruf vererbt wurde, wird zum Ausnahmecode bei vererbten Ausnahmen der Wert 100 000 addiert.
Triviale Ausnahmen koennen nicht aus einer Subroutine oder Funk-

tion heraus zu deren Aufruf vererbt werden. Werden sie nicht durch eine eigene Ausnahmeroutine behandelt, erfolgt in jedem Fall die Standardbehandlung.
Mit der cause-Anweisung koennen Ausnahmen ausgeloest werden um z.B. Ausnahmeroutinen zu testen oder vom Programm erkannte irregulaere Zustaende mit Ausnahmeroutinen zu behandeln.
Der numerische Ausdruck in der cause-Anweisung liefert den Ausnahmecode. Fuer die Kennzeichnung "privater" Ausnahmen sind die Ausnahmecodes von 1 bis 999 vorgesehen. Alle uebrigen Ausnahmecodes sind entweder bereits belegt (s. Anl.3.) oder fuer Erweiterungen des BASIC-Systems reserviert.

Beispiel

Die folgende Funktion berechnet den reziproken Wert ihres numerischen Arguments. Entsteht dabei ein Ueberlauf, wird als Ergebnis der groesste darstellbare Wert (MAXNUM) geliefert, bei Unterlauf wird Null geliefert.

```
750  FUNCTION RPW (arg)
760  WHEN EXCEPTION IN
770       LET RPW = 1/arg
780  USE
790       IF EXTYPE = 1002 OR EXTYPE = 3001 THEN
800       LET RPW = MAXNUM
810       ELSE
820       LET RPW = 0 ! nur noch Unterlauf moeglich
830  END IF
840  END WHEN
850  END FUNCTION
```

Das folgende Programmstueck ist ein Beispiel fuer die Fehlerbehandlung beim Eroeffnen einer Datei.
Es werden Ausnahmen mit den Ausnahmecodes zwischen 7100 und 7199 behandelt. Alle uebrigen Fehler werden abgewiesen. Als Fehlerbehandlung wird das Wiederholen der open-Anweisung, das Fortsetzen mit nicht eroeffneter Datei und das Eroeffnen einer anderen Datei vorgesehen.

```
.
.
.
1410 LET datei_offen$ = "nein"
1420 LET datei_name$ = "ausgabe2"
1430 WHEN EXCEPTION USE dateifehler
1440 OPEN #1: NAME datei_name$, ACCESS INPUT
1450 LET datei_offen$ = "ja"
1460 END WHEN
.
.
.
```

```
1700 HANDLER dateifehler
1710  IF EXTYPE < 7100 OR EXTYPE >= 7200 THEN &
&      EXIT HANDLER
1720  PRINT "Fehler bei Dateieroeffnung"
1730  PRINT "Fehler: ";EXTYPE;"ZEILE: ";EXLINE;"Datei:"&
&      ; datei name□
1740  PRINT"(1)Wiederholen (2)neue Datei (3)nicht eroeffnen"
1750  INPUT PROMPT "Auswahl: ":auswahl
1760  SELECT CASE auswahl
1770   CASE 1
1780    RETRY
1790   CASE 2
1800    INPUT PROMPT "neuer Dateiname:" datei_name□
1810    RETRY
1820   CASE 3
1830   CASE ELSE
1840    GOTO 1740
1850  END SELECT
1860 END HANDLER
```

12.2. Testunterstuetzung

In ein BASIC-Programm koennen Anweisungen zur Testunterstuetzung aufgenommen werden. Mit der trace-Anweisung wird die Protokollierung von Wertaenderungen von Variablen sowie Verzweigungen eingeschaltet.
Die break-Anweisung unterbricht den Programmlauf.
Die debug-Anweisung steuert einen Teststatus, von dem es abhaengt, ob trace- und break-Anweisungen ausgefuehrt oder uebergangen werden.

Syntax

```
debug_anweisung ::= DEBUG <[ ON  ]>
                          [ OFF ]

break_anweisung ::= BREAK

trace_anweisung ::= TRACE <[ ON [ kanal_ausdruck] ]>
                          [ OFF                   ]

kanal_ausdruck ::= *index

index ::= num_ausdruck
```

Jede Programmeinheit verfuegt ueber einen Teststatus und einen Protokollstatus. Beide koennen jeweils "aktiv" oder "nicht aktiv" sein. Zu Beginn der Programmabarbeitung sind Test- und Protokollstatus jeder Programmeinheit nicht aktiv.
Durch die Abarbeitung einer debug-on-Anweisung wird der Teststatus fuer die entprechende Programmeinheit aktiv gesetzt.

Der Teststatus bleibt dann von Aufruf zu Aufruf dieser Programmeinheit aktiv und kann nur durch eine debug-off-Anweisung "nicht aktiv" gesetzt werden.
Die Abarbeitung einer trace-on-Anweisung haengt vom Teststatus ab. Bei aktivem Teststatus wird durch eine trace-on-Anweisung der Protokollstatus aktiv gesetzt. Bei nicht aktivem Teststatus ist die trace-on-Anweisung wirkungslos.
Bei aktivem Protokollstatus werden Wertaenderungen von Variablen und Verzweigungen protokolliert. Bei der Wertaenderung einer Variablen werden die Zeilennummer der Anweisung auf der die Wertaenderung erfolgte, der neue Wert und bei indizierten Variablen die Indizes protokolliert. Bei jeder Verzweigung wird die Zeilennummer der Anweisung protokolliert, die die Verzweigung bewirkt, und die Zeilennummer, zu der verzweigt wird.
Das Protokoll wird ueber den in der trace-Anweisung angegebenen Kanal ausgegeben oder ueber Terminal, falls kein Kanal angegeben war. Bei der Protokollausgabe ueber einen Kanal muss dieser Kanal mit einer Ausgabedatei verbunden sein. Diese Datei muss mit den Angaben ORGANIZATION SEQUENTIAL, RECTYPE DISPLAY und ACCESS OUTPUT oder ACCESS OUTIN eroeffnet werden.
Die trace-off-Anweisung setzt den Protokollstatus "nicht aktiv" und beendet eine evtl. laufende Protokollausgabe. Im Unterschied zum Teststatus bleibt der Protokollstatus nicht von Aufruf zu Aufruf einer Programmeinheit unveraendert, sondern ist bei jedem Aufruf einer Programmeinheit zunaechst nicht aktiv.
Es ist zu beachten, dass eine debug-off-Anweisung den Protokollstatus nicht beeinflusst. Eine laufende Protokollierung wird durch eine debug-off-Anweisung nicht beendet. Sie laeuft weiter, bis eine trace-off-Anweisung abgearbeitet oder die Programmeinheit verlassen wird.
Die Abarbeitung einer break-Anweisung haengt ebenfalls vom Teststatus ab. Ist der Teststatus aktiv, loest die Abarbeitung der break-Anweisung eine triviale Ausnahme aus. Ist der Teststatus nicht aktiv, ist die break-Anweisung wirkungslos. Die Standardbehandlung einer durch die break-Anweisung ausgeloesten trivialen Ausnahme besteht in der Ausgabe der Zeilennummer der break-Anweisung und der anschliessenden Unterbrechung des Programmlaufs. Das unterbrochene Programm kann mit der Testhilfe weiter bearbeitet werden (siehe Anleitung fuer den Bediener).

Beispiel

Das folgende Programmstueck protokolliert Teile einer Reihenentwicklung, falls diese nicht in 100 Schritten konvergiert und unterbricht die Abarbeitung; falls ein Ergebnis auch nach 150 Schritten nicht vorliegt.

```
.
.
.
1000 EXTERNAL FUNCTION P4
1010 ! Berechnung von PI/4 durch 1-(1/3) + (1/5) - (1/7)
1020 DEBUG ON
1030 LET schrittzahl = 0
1040 LET w, signum, fwert = 1
1060 DO
1070   LET schrittzahl = schrittzahl + 1
1080   IF schrittzahl > 100 THEN TRACE ON
1090   IF schrittzahl > 150 THEN BREAK
1100   LET w = w + 2
1110   LET signum = -signum
1120   LET quot = 1/w
1130   IF quot < EPS (fwert) THEN
1140         LET p4 = fwert
1150         EXIT FUNCTION
1160   END IF
1170   LET fwert = fwert + w * signum
1180 LOOP
1190 END FUNCTION
```

Anlage 1: Erweiterungen

Im Abschnitt 5. wurde gezeigt, wie numerische Werte definiert und verarbeitet werden koennen. Es wurde darauf hingewiesen, dass alle numerischen Werte dezimale Gleitkommazahlen sind. Innerhalb einer Programmeinheit kann mit Hilfe einer Arithmetik-Option (s. Abschn. 5.6.) festgelegt werden, ob numerische Werte mit einfacher oder doppelter Genauigkeit dargestellt werden sollen. Daraus folgt, dass innerhalb einer Programmeinheit (Hauptprogramm oder externe Prozedur) immer nur ein numerischer Datentyp existiert und alle numerischen Groessen von diesem Typ sind.
Diese Festlegung entspricht genau der Festlegung des zugrunde liegenden Standards ANSI X3.113 1987. Viele andere BASIC-Versionen und andere hoehere, problemorientierte Programmiersprachen lassen innerhalb eines Programms bzw. einer Programmeinheit gleichzeitig verschiedene numerische Datentypen zu. Insbesondere ist es fuer eine effektive Realisierung vieler Algorithmen vorteilhaft, mit ganzzahligen Werten (Integer-Werten) zu arbeiten. Zum Beispiel muessen die Indizes von indizierten Variablen stets ganzzahlig sein. Hier ist es besser, von vornherein mit ganzzahligen Werten und nicht mit gerundeten Gleitkommazahlen zu arbeiten.
Um dem Nutzer die Moeglichkeit zu geben, seine Programme durch Verwendung von Integer-Werten effektiver zu gestalten, wurde eine Erweiterung des Standards vorgenommen. Die Ausdrucksmittel fuer diese Erweiterung werden in einem sogenannten Vorspiel zusammengefasst.

Syntax

```
programm ::=
  [vorspiel] hauptprogramm [{kommentarzeile}* externe_prozedur]*

vorspiel ::=
                  ┌                          ┐
                  │ kommentarzeile           │*
  prelude_zeile <│                          │> prelude_end_zeile
                  │ prelude_option_zeile     │
                  └                          ┘

prelude_zeile ::= zeilennummer PRELUDE zeilenrest

prelude_end_zeile ::= zeilennummer END PRELUDE zeilenrest

prelude_option_zeile ::= zeilennummer prelude_option zeilenrest
                          ┌                          ┐
                          │ integer_spezifikation    │
prelude_option::= OPTION <│                          │>
                          │ grenzwert_spezifikation  │
                          └                          ┘

                                                      *
integer_spezifikation ::= INTEGER {num_identifikator},

grenzwert_spezifikation ::= BOUNDS [NOT] INTEGER
```

Soll die Integer-Erweiterung benutzt werden, so muss vor dem Hauptprogramm ein Vorspiel stehen. Dieses Vorspiel enthaelt Anweisungen, die die Arbeit mit Integer-Groessen ermoeglichen und die nicht im Standard ANSI X3.113 1987 enthalten sind. Jedes Programm, das ein Vorspiel enthaelt, kann durch Streichen dieses Vorspiels in ein standardgerechtes Programm gemaess den Festlegungen des ANSI-Standards ueberfuehrt werden.
Integer-Werte sind ganzzahlige numerische Werte im Wertebereich von -32768 bis +32767.
Ist im Vorspiel eine Integer-Spezifikation angegeben, so besitzen alle Objekte, die in allen folgenden Programmeinheiten durch einen in der Integer-Spezifikation aufgefuehrten Identifikator bezeichnet werden, den Datentyp Integer. Wird also ein solcher Identifikator zur Bezeichnung einer einfachen Variablen verwendet, so kann diese Variable nur Integer-Werte enthalten. Wird er zur Bezeichnung eines Feldes verwendet, so koennen alle Feldelemente nur Integer-Werte enthalten. Wird er zur Bezeichnung einer Funktion verwendet, so liefert diese Funktion einen Integer-Wert als Resultat.
Durch eine Integer-Spezifikation im Vorspiel wird also bestimmten Identifikatoren die Eigenschaft zugeordnet, ausschliesslich Integer-Objekte zu bezeichnen. Alle weiteren Festlegungen ueber die Verwendung von Identifikatoren bleiben erhalten.
Ist im Vorspiel eine Integer-Spezifikation angegeben, so werden alle ganzzahligen numerischen Konstanten im Programm, die im Bereich der Integer-Werte liegen, auch als Integer-Werte dargestellt.
Im folgenden Beispiel soll ein Programm aus einem Hauptprogramm und einer externen up-Definition bestehen. In beiden Programmeinheiten sollen die einfachen Integer-Variablen I und J lokal verwendet werden. Weiterhin soll im Hauptprogramm eine Matrix M von Integer-Werten definiert und u. a. als Parameter an einen Ruf des externen Subroutine uebergeben werden. In der externen Subroutine soll eine interne Funktion F definiert und verwendet werden, die als Resultat einen Integer-Wert liefert. Dann muss dem Programm folgendes Vorspiel vorangestellt werden. Zur Illustration sind auch das Hauptprogramm und das externe Unterprogramm angedeutet.

```
10   PRELUDE
20   OPTION INTEGER I,J,M,PM,F
30   END PRELUDE
  .
  .
  .
100  PROGRAM BEISPIEL
110  DIM M(1 TO 5, 0 TO 3)
120  DECLARE EXTERNAL SUB UP
 .
 .
 .
180  LET X = M(I,J)+4.5
190  CALL UP(X,M)
 .
 .
 .
300  END
400  EXTERNAL SUB UP(PX,PM)
410  FUNCTION F(A¤)
 .
 .
 .
450  END FUNCTION
 .
 .
 .
510 LET PM(I,J) = F(X¤ & Y¤)
 .
 .
 .
600  END SUB
```

Es sei nochmals darauf hingewiesen, dass in dem obigen Programm die I,J,M,PM und F stets Integer-Objekte bezeichnen. Sie koennen also nicht zur Bezeichnung einer Groesse in Gleitkommadarstellung verwendet werden. Durch die Integer-Erweiterung tritt der Fall auf, dass innerhalb einer Programmeinheit numerische Groessen zweier Typen (Gleitkomma einfacher bzw. doppelter Genauigkeit, Integer) gemischt auftreten koennen. In einem solchen Fall sind einige Besonderheiten zu beachten:
Besitzt eine arithmetische Operation zwei Operanden unterschiedlichen Typs, so hat das Resultat der Operation den Typ gemaess der folgenden Tabelle:

Tabelle 7: Typ des Resultates einer arithmetischen Operation

		rechter Operand	
		integer	Gleitkomma
linker Operand	integer	integer	Gleitkomma
	Gleitkomma	Gleitkomma	Gleitkomma

Insbesondere ist zu beachten, dass die Division eine ganzzahlige Division ist, sofern beide Operanden Integer-Groessen sind. Bezeichnet z.B. I eine Integer-Variable, so liefert

```
I/4
```

den Wert 2, falls I den Wert 10 besitzt. Fehlt das Vorspiel, so wuerde der obige Ausdruck den Wert 2.5 liefern. Andererseits kann durch

```
I/4.0
```

erreicht werden, dass das Ergebnis immer in Gleitkommadarstellung ermittelt wird. Da 4.0 eine Konstante in Gleitkommadarstellung ist, wird der Wert von I in die Gleitkommadarstellung konvertiert, falls I eine Integer-Variable ist.
Auch bei der Potenzierung wird stets ein Integer-Ergebnis ermittelt, wenn Basis und Exponent Integer-Werte sind.

Sind beide Operanden einer arithmetischen Operation Integer-Operanden, so wird ein Ueberlauf gemeldet, wenn das Ergebnis nicht mehr im Wertebereich fuer Integer-Werte liegt. So liefert

```
I * 1000
```

einen Ueberlauf, falls I eine Integer-Variable ist und den Wert 40 besitzt. Dagegen liefert

```
I * 1000.0
```

den Wert 40000, da durch die Gleitkommakonstante 1000.0 eine Konvertierung des Wertes von I in die Gleitkommadarstellung erzwungen wird.
Muss an eine Integer-Variable ein Gleitkommawert zugewiesen werden, so wird dieser Wert zunaechst gerundet und anschliessend in einen Integer-Wert konvertiert. Dabei kann ein Ueberlauf entstehen.
Enthaelt eine numerische LET-Anweisung mehrere Zielvariable, so muessen alle den gleichen Typ besitzen. Die Anweisung

```
LET A,I,K1 = 20.5
```

ist also nur korrekt, falls A,I und K1 entweder Variable des Typs Integer oder Variable des Typs Gleitkomma sind. Falls A,I und K1 vom Typ integer sind, wird ihnen der Wert 21 zugeordnet.
Bei der Uebergabe von numerischen Parametern an Funktionen und Subroutinen werden notwendige Anpassungen automatisch vorgenommen, sofern die Parameteruebergabe "by value" erfolgt (s. Abschn. 9.1.3.). Erfolgt die Parameteruebergabe "by reference" (s. Abschn. 9.2.3.), so muss gesichert werden, dass aktuelle und formale Parameter den gleichen numerischen Typ besitzen.

Integer-Werte sollten aus Effektivitaetsgruenden ueberall dort verwendet werden, wo durch den Algorithmus von vornherein klar ist, dass bestimmte Groessen nur ganzzahlige Werte annehmen koennen. Ein typisches Beispiel dafuer sind die Indizes in indizierten Variablen. Sei z.B. die Laufanweisung

```
100  FOR I=1 TO 10
110    LET A(I) = B(I)+C(I)
120  NEXT I
```

betrachtet. Die Variable I kann hier nur ganzzahlige Werte annehmen. In einem Programm, das die Integer-Erweiterung nicht benutzt, ist die Variable I eine Gleitkommavariable (in der Regel von doppelter Genauigkeit). Alle Operationen in der Laufanweisung, die die Laufvariable I betreffen (Erhoehen um die Schrittweite, Endetest, Berechnung der Adresse der indizierten Variablen), erfolgen dann im Gleitkommaformat und werden gerundet. Es ist klar, dass die Abarbeitung der Laufanweisung wesentlich effektiver erfolgen kann, wenn die Variable I den Typ Integer besitzt.
Der Nutzer sollte also stets pruefen, ob in seinem Algorithmus Groessen vorkommen, die nur ganzzahlige Werte im Bereich der Integer-Werte annehmen koennen. In einem solchen Fall sollte aus Effektivitaetsgruenden die Integer-Erweiterung genutzt werden.

Es wurde bereits darauf hingewiesen, dass die Verwendung von Integer-Variablen als Indizes fuer indizierte Variable Vorteile bei der Adressberechnung mit sich bringt. In diesem Zusammenhang ist jedoch noch ein weiteres Problem zu beachten.
Bei der Deklaration von Feldern muessen Indexgrenzen vorgegeben werden (s. Abschn. 7.1.). Diese Indexgrenzen muessen ganzzahlig sein, jedoch ist ihre Groesse nicht beschraenkt. Das heisst, das die beiden folgenden Felder jeweils 10 Elemente enthalten.

```
120  DIM A(1 TO 10), X(100001 TO 100010)
```

Waehrend fuer das Feld A die Indexgrenzen Integer-Werte sind, muessen die Indexgrenzen fuer das Feld X intern durch Gleitkommazahlen dargestellt werden. Da A durch Redimensionierung zur Laufzeit des Programms Indexgrenzen annehmen kann, die wie bei X nicht durch Integer-Werte dargestellt werden koennen, muessen die Indexgrenzen generell als Gleitkommazahlen angesehen werden.
Bei der Verarbeitung eines Feldes werden die Indexgrenzen benoetigt, da fuer jeden Index einer indizierten Variablen zur Programmlaufzeit ueberprueft wird, ob er innerhalb der Indexgrenzen liegt. Dieser Vergleich muss in Gleitkommaformat erfolgen, da die Indexgrenzen intern Gleitkommazahlen sind. Dies ist eine weitere Uneffektivitaet, wenn man bedenkt, dass fuer die Mehrzahl der praktischen Faelle die Indexgrenzen auch als Integer-Werte dargestellt werden koennten und demzufolge der oben erwaehnte Test der Indizes im Integer-Format erfolgen wuerde. Um dies zu ermoeglichen, kann eine Grenzwert-Spezifikation im Vorspiel angegeben werden.
Ist im Vorspiel die Spezifikation

```
    OPTION BOUNDS INTEGER
```

angegeben, so muessen alle Felder, die in den folgenden Programmeinheiten definiert werden, Indexgrenzen besitzen, die im Bereich der Integer-Werte (-32768 bis +32767) liegen. Auch bei Redimensionierungen muessen die neu eingestellten Indexgrenzen in diesem Bereich liegen. Wird eine Indexgrenze verwendet, die ausserhalb dieses Bereiches liegt, so wird ein Fehler gemeldet. Ist im Vorspiel die Spezifikation

```
    OPTION BOUNDS NOT INTEGER
```

angegeben, so koennen bei der Definition von Feldern beliebige Indexgrenzen verwendet werden. Diese Spezifikation kann auch entfallen, da sie der Standardannahme entspricht.
Gilt die Grenzwertspezifikation OPTION BOUNDS INTEGER, so ist

eine sehr effektive Verarbeitung von Feldern moeglich, sofern gleichzeitig durch eine Integer-Spezifikation die Moeglichkeit geschaffen wird, Integer-Variable als Indizes fuer indizierte Variable zu verwenden. Deshalb sollten beide Spezifikationen verwendet werden, wenn dies moeglich ist.

Das folgende Beispiel illustriert nochmals die Verwendung des Vorspiels. Ueber Terminal werden zwei Vektoren elementweise eingegeben. Anschliessend wird das Skalarprodukt gebildet und ausgegeben.

```
10   PRELUDE
20   OPTION INTEGER ANZAHL,I,N
30   OPTION BOUNDS INTEGER
40   END PRELUDE
100  PROGRAM SKALARES_PRODUKT
110  OPTION BASE 1
120  DIM X(100),Y(100)
200  SUB EINGABE(VEKTOR(),ANZAHL)
210  FOR I=LBOUND(VEKTOR) TO LBOUND(VEKTOR)+ANZAHL-1
220    INPUT PROMPT STR□(I)&". ZAHL = ": VEKTOR(I)
230  NEXT I
240  END SUB
300  INPUT PROMPT "ANZAHL DER ELEMENTE EINES VEKTORS = ":N
310  IF 1 <= N AND N <= 100 THEN 340
320  PRINT "FEHLERHAFTE ANZAHL"
330  STOP
340  PRINT "EINGABE VEKTOR X"
350  CALL EINGABE(X,N)
360  PRINT "EINGABE VEKTOR Y"
370  CALL EINGABE(Y,N)
380  LET SKALARPRODUKT = 0
390  FOR I = 1 TO N
400    LET SKALARPRODUKT = SKALARPRODUKT + X(I) * Y(I)
410  NEXT I
420  PRINT "SKALARPRODUKT = "; SKALARPRODUKT
430  END
```

Anlage 2: Zeichensatz

Position	Code hexadezimal	Code oktal	grafische Darstellung	ORD Mnemonik	Name
0.	00	0		NUL	Null
1.	01	1		SOH	Start of heading
2.	02	2		STX	Start of text
3.	03	3		ETX	End of text
4.	04	4		EOT	End of transmission
5.	05	5		ENQ	Enquiry
6.	06	6		ACK	Acknowledge
7.	07	7		BEL	Bell
8.	08	10		BS	Backspace
9.	09	11		HT	Horizontal tab
10.	0A	12		LF	Line feed
11.	0B	13		VT	Vertical tab
12.	0C	14		FF	Form feed
13	0D	15		CR	Carriage return
14.	0E	16		SO	Shift out
15.	0F	17		SI	Shift in
16.	10	20		DLE	Data link escape
17.	11	21		DC1	Device control 1
18.	12	22		DC2	Device control 2
19.	13	23		DC3	Device control 3
20.	14	24		DC4	Device control 4
21.	15	25		NAK	Negative acknowledge
22.	16	26		SYN	Synchronous idle
23.	17	27		ETB	End of trans block
24.	18	30		CAN	Cancel
25.	19	31		EM	End of medium
26.	1A	32		SUB	Substitute
27.	1B	33		ESC	Escape
28.	1C	34		FS	File separator
29.	1D	35		GS	Group separator
30.	1E	36		RS	Record separator
31.	1F	37		US	Unit separator
32.	20	40		SP	Space
33.	21	41	!		
34.	22	42	"		
35.	23	43	#		
36.	24	44	$		

Anlage 2: Fortsetzung

37.	25	45	%		
38.	26	46	&		
39.	27	47	'		
40.	28	50	(		
41.	29	51	)		
42.	2A	52	*		
43.	2B	53	+		
44.	2C	54	,		
45.	2D	55	-		
46.	2E	56	.		
47.	2F	57	/		
48.	30	60	0		
49.	31	61	1		
50.	32	62	2		
51.	33	63	3		
52.	34	64	4		
53.	35	65	5		
54.	36	66	6		
55.	37	67	7		
56.	38	70	8		
57.	39	71	9		
58.	3A	72	:		
59.	3B	73	;		
60.	3C	74	<		
61.	3D	75	=		
62.	3E	76	>		
63.	3F	77	?		
64.	40	100	@		
65.	41	101	A		
66.	42	102	B		
67.	43	103	C		
68.	44	104	D		
69.	45	105	E		
70.	46	106	F		
71.	47	107	G		
72.	48	110	H		
73.	49	111	I		
74.	4A	112	J		
75.	4B	113	K		
76.	4C	114	L		
77.	4D	115	M		
78.	4E	116	N		
79.	4F	117	O		
80.	50	120	P		
81.	51	121	Q		
82.	52	122	R		
83.	53	123	S		
84.	54	124	T		
85.	55	125	U		
86.	56	126	V		
87.	57	127	W		
88.	58	130	X		
89.	59	131	Y		
90.	5A	132	Z		
91.	5B	133	[		
92.	5C	134	\		
93.	5D	135	]		

Anlage 2: Fortsetzung

94.	5E	136	^		
95.	5F	137	_	UND	
96.	60	140	`	GRA	
97.	61	141	a	LCA	
98.	62	142	b	LCB	
99.	63	143	c	LCC	
100.	64	144	d	LCD	
101.	65	145	e	LCE	
102.	66	146	f	LCF	
103.	67	147	g	LCG	
104.	68	150	h	LCH	
105.	69	151	i	LCI	
106.	6A	152	j	LCJ	
107.	6B	153	k	LCK	
108.	6C	154	l	LCL	
109.	6D	155	m	LCM	
110.	6E	156	n	LCN	
111.	6F	157	o	LCO	
112.	70	160	p	LCP	
113.	71	161	q	LCQ	
114.	72	162	r	LCR	
115.	73	163	s	LCS	
116.	74	164	t	LCT	
117.	75	165	u	LCU	
118.	76	166	v	LCV	
119.	77	167	w	LCW	
120.	78	170	x	LCX	
121.	79	171	y	LCY	
122.	7A	172	z	LCZ	
123.	7B	173	{	LBR	
124.	7C	174	|	VLN	
125.	7D	175	}	RBR	
126.	7E	176	~	TIL	
127.	7F	177		DEL	Delete

Anlage 3: Ausnahmecodes

Jeder Ausnahme, die waehrend der Programmabarbeitung auftritt, wird ein Ausnahmecode zugeordnet. Wird die Standardbehandlung fuer Ausnahmen wirksam, so wird dieser Ausnahmecode dem Nutzer neben anderen Informationen ueber das Bedienterminal mitgeteilt. Innerhalb von Ausnahmeroutinen kann der Ausnahmecode ueber die Standardfunktion EXTYPE ermittelt werden, das heisst diese Funktion liefert als Resultat einen Ausnahmecode.
Die folgende Uebersicht enthaelt alle Ausnahmecodes, die bei der Programmabarbeitung auftreten koennen, einschliesslich einer Beschreibung der Ursache ihres Auftretens. Steht vor dem Ausnahmecode ein Ausrufezeichen, so handelt es sich um eine triviale Ausnahme. Sonst ist die Ausnahme nicht trivial. Die in Klammern angegebenen Nummern sind die Nummern der Abschnitte, in denen auf die Ausnahme hingewiesen wurde. Bei trivialen Ausnahmen wird die Art der Behandlung beschrieben.
Bei Ausnahmen, die von Funktionen oder Prozeduren an die rufende Programmeinheit vererbt werden (s. Abschn. 12.1.), wird ein Ausnahmecode bereitgestellt, der sich aus einem der unten aufgefuehrten Codes plus 100000 ergibt.

Ueberlaeufe (1000)

1000 nicht spezifizierbarer arithmetischer Fehler

-1001 Integer-Ueberlauf (s. Anl. 1)

1002 Ueberlauf bei der Berechnung eines numerischen Ausdrucks (s. Abschn. 5.3.)

1003 Ueberlauf beim Resultat einer numerischen Standardfunktion (s. Abschn. 5.4)

1004 Ueberlauf beim Resultat der Standardfunktion VAL (s. Abschn. 6.4.)

1006 Ueberlauf bei einem numerischen Datum in der internen Datenliste (s. Abschn. 10.1)

! 1007 Ueberlauf bei einem numerischen Datum, das ueber Terminal eingegeben wurde (s. Abschn. 10.2.)
Behandlung: interaktive Fehlerkorrektur durch den Bediener

1008 Ueberlauf bei einem numerischen Datum, das ueber eine Datei (nicht Terminal) eingegeben wurde (s. Abschn. 11.4.)

1051 Ueberlauf bei der Berechnung eines string-Ausdrucks (s. Abschn. 6.3.)

1053 Ueberlauf bei der Zuweisung eines aus der internen Datenliste gelesenen string-Wertes an eine Variable (s. Abschn. 10.1.)

! 1054 Ueberlauf bei der Zuweisung eines ueber Terminal gelesenen string-Wertes an eine Variable (s. Abschn. 10.2.)
Behandlung: interaktive Fehlerkorrektur durch den Bediener

1105 Ueberlauf bei der Zuweisung eines ueber eine Datei (nicht Terminal) gelesenen string-Wertes an eine Variable (s. Abschn. 11.4.)

1106 Ueberlauf bei der Zuweisung eines string-Wertes an eine Variable durch eine string-let- bzw. eine string-deflet-Anweisung (s. Abschn. 6.5. und 9.1.)

Unterlaeufe (1500)

Alle Unterlaeufe sind triviale Ausnahmen. Die Behandlung besteht in der Annahme des Wertes Null und der Fortsetzung der Programmabarbeitung.

! 1502 Unterlauf bei der Berechnung eines numerischen Ausdrucks (s. Abschn. 5.1.)
! 1503 Unterlauf beim Resultat einer numerischen Standardfunktion (s. Abschn. 5.4.)
! 1504 Unterlauf beim Resultat der Standardfunktion VAL (s. Abschn. 6.4.)
! 1506 Unterlauf bei einem numerischen Datum in der internen Datenliste (s. Abschn. 10.1.)
! 1507 Unterlauf bei einem numerischen Datum, das ueber Terminal eingegeben wurde (s. Abschn. 10.2.)
! 1508 Unterlauf bei einem numerischen Datum, das ueber eine Datei (nicht Terminal) eingegeben wurde (s. Abschn. 11.4.)

Indexfehler (2000)

2001 Der Index einer indizierten Variablen liegt ausserhalb des zulaessigen Bereiches (s. Abschn. 5.2. und 6.2.).

Mathematische Fehler (3000)

3001 Division durch Null (s. Abschn. 5.3.)
3002 Potenzierung einer gebrochenen Basis mit einem negativen Exponenten (s. Abschn. 5.3.)
3003 Potenzierung der Basis Null mit einem negativen Exponenten (s. Abschn. 5.3.)
3004 Das Argument einer Logarithmusfunktion (LOG, LOG10, LOG2) ist Null oder negativ (s. Abschn. 5.4.).
3005 Das Argument der Wurzelfunktion (SQR) ist negativ (s. Abschn. 5.4.).
3006 Das zweite Argument der Standardfunktion MOD bzw. REMAINDER (der Divisor) ist Null (s. Abschn. 5.4.).
3007 Das Argument der Standardfunktion ACOS bzw. ASIN liegt nicht im Bereich -1 <= x <= 1 (s. Abschn. 5.4.).
3008 Beide Argumente der Standardfunktion ANGLE sind Null (s. Abschn. 5.4.).

Initialisierungsfehler (3100)

! 3102 Eine string-Variable wird verwendet, bevor ihr ein Wert zugewiesen wurde (s. Abschn. 6.2.).
Behandlung: Zuweisung eines leeren string-Wertes und Fortsetzung der Programmabarbeitung

Parameterfehler (4000)

4001 Das Argument der Standardfunktion VAL ist keine numerische Konstante (s. Abschn. 6.4.).
4002 Das Argument der Standardfunktion CHR□ liegt ausserhalb des zulaessigen Bereiches (s. Abschn. 6.4.).

4003 Das Argument der Standardfunktion ORD ist weder ein gueltiges Zeichen noch ein gueltiges Mnemonik (s. Abschn. 6.4.).

4004 Bei der Standardfunktion SIZE wird ein Index angegeben, der ausserhalb des zulaessigen Bereiches liegt (s. Abschn. 7.1.).

! 4005 Der Index der Tabulator-Funktion TAB ist kleiner als 1 (s. Abschn. 10.3.).
Behandlung: Annahme von TAB(1) und Fortsetzung der Programmabarbeitung

4006 Es wird versucht, die Zeilenbreite bzw. die aktuelle Satzlaenge auf einen Wert einzustellen, der kleiner als die Zonenlaenge bzw. groesser als die maximale Satzlaehge ist (s. Abschn. 10.3., 11.3.).

4007 Es wird versucht, die Zonenlaenge auf einen Wert einzustellen, der kleiner als 1 bzw. groesser als die Zeilenbreite (aktuelle Satzlaenge) ist (s. Abschn. 10.3., 11.3.).

4008 Bei der Standardfunktion LBOUND wird ein Index angegeben, der ausserhalb des zulaessigen Bereiches liegt (s. Abschn. 7.1.).

4009 Bei der Standardfunktion UBOUND wird ein Index angegeben, der ausserhalb des zulaessigen Bereiches liegt (s. Abschn. 7.1.).

4010 Das zweite Argument der Standardfunktion REPEAT$ ist kleiner als Null (s. Abschn. 6.4.).

4301 Die Anzahl bzw. der Typ der Argumente in einer chain-Anweisung stimmen nicht mit der Anzahl bzw. den Typen der Funktionsparameter in der Programmzeile des gerufenen Programms ueberein (s. Abschn. 9.3.).

4302 Die Anzahl der Dimensionen eines Feldes als Argument in einer chain-Anweisung stimmt nicht mit der Anzahl der Dimensionen des entsprechenden Funktionsparameters in der Programmzeile des gerufenen Programms ueberein (s. Abschn. 9.3.).

4303 Durch eine chain-Anweisung werden numerische Werte oder Felder als Argumente vermittelt und das rufende und das gerufene Programm besitzen unterschiedliche Arithmetik-Options bzw. unterschiedliche Grenzwert-Spezifikationen (s. Abschn. 9.3. und Anlage 1).

Dateifehler (7000)

7001 Die Kanalnummer liegt nicht im Bereich von 1 bis 99 (s. Abschn. 11.1.).

! 7002 Der Kanal 0 wird in einer open-, close- oder erase-Anweisung angegeben bzw. es wird versucht, den Dateizeiger zu positionieren (s. Abschn. 11.1., 11.2., 11.3. und 11.4.).
Behandlung: Fortsetzung der Programmabarbeitung

7003 In einer open-Anweisung wird ein Kanal ungleich Kanal 0 spezifiziert, der bereits aktiv ist (s. Abschn. 11.1.).

7004 In einer Datei- bzw. Satzoperation ausser der open- oder der ask-Anweisung wird ein Kanal spezifiziert, der nicht aktiv ist (s. Abschn. 11.1., 11.2., 11.3., 11.4.).

7051 In einer open-Anweisung wird eine maximale Satzlaenge angegeben, die nicht groesser als Null ist (s. Abschn. 11.1.).

7100 In einer open-Anweisung wird fuer ein Dateiattribut ein fehlerhafter Wert angegeben (s. Abschn. 11.1.).

7101 Es wird versucht, mehr als 4 Dateien (ausser Kanal 0) gleichzeitig zu eroeffnen (s. Abschn. 11.1.).

7102 Es wird versucht, eine nicht vorhandene Datei als Eingabedatei zu eroeffnen, bzw. es wird eine nicht korrekte Pfadangabe angegeben (s. Abschn. 11.1.).

7103 Die Pfadangabe ist fehlerhaft (s. Abschn. 11.1.).

7104 Es wird versucht, mehr Dateien gleichzeitig zu oeffnen, als dies vom Betriebssystem unterstuetzt wird (s. Abschn. 11.1.).

7105 Bei der Bereitstellung des Speichers, der fuer die Arbeit mit einer neu zu eroeffnenden Datei benoetigt wird, tritt ein Ueberlauf des zur Verfuegung stehenden Speichers auf (s. Abschn. 11.1.).

7111 Eine Datei soll mit der Dateiorganisation STREAM und der Satzart DISPLAY eroeffnet werden (s. Abschn. 11.1.).

7112 Es wird versucht ein Geraet mit der Satzart INTERNAL zu eroeffnen (s. Abschn. 11.1.)

7113 Die eroeffnete Datei laesst die gewaehlte Zugriffsart nicht zu (s. Abschn. 11.1.).

7204 Die Zeigerpositionierung SAME ist gefordert, jedoch wurde zuvor die open-Anweisung bzw. eine fehlerhafte Datei- oder Satzoperation ausgefuehrt (s. Abschn. 11.2., 11.3. und 11.4.).

! 7205 Es wird versucht, eine Zeigerpositionierung auf ein Geraet anzuwenden (s. Abschn. 11.2., 11.3. und 11.4.).
Behandlung: Fortsetzung der Programmabarbeitung

7301 Eine erase-Anweisung wird auf eine Datei angewandt, die nicht mit der Zugriffsart OUTIN eroeffnet wurde (s. Abschn. 11.1.).

7302 Es wird versucht, in eine Datei zu schreiben, die mit der Zugriffsart INPUT eroeffnet wurde (s. Abschn. 11.3.).

7303 Es wird versucht, aus einer Datei zu lesen, die mit der Zugriffsart OUTPUT eroeffnet wurde (s. Abschn. 11.4.).

7305 Es wird versucht, einen nicht existierenden Datensatz einzulesen (Datenexistenzbedingung ist falsch) (s. Abschn. 11.4.).

7308 Es wird versucht, einen existierenden Datensatz zu ueberschreiben (Datenexistenzbedingung ist wahr) (s. Abschn. 11.3.).

! 7311 Die erase-Anweisung wird auf ein Geraet angewandt, das nicht die Faehigkeit zum Loeschen von Daten besitzt (s. Abschn. 11.1.).
Behandlung: Fortsetzung der Programmabarbeitung

7312 Es wird versucht, die aktuelle Satzlaenge bzw. die Zonenlaenge fuer eine Datei zu setzen, die nicht die Satzart DISPLAY besitzt (s. Abschn. 11.3.).

7313 Es wird versucht, die aktuelle Satzlaenge, bzw. die Zonenlaenge fuer eine Datei zu setzen, die die Zugriffsart INPUT besitzt (s. Abschn. 11.3.).

7317 Eine print-Anweisung wird fuer eine Datei verwendet, die die Satzart INTERNAL besitzt (s. Abschn. 11.3.).

7318 Eine input- oder line-input-Anweisung wird fuer eine Datei verwendet, die die Satzart INTERNAL besitzt (s. Abschn. 11.3.).

7321 Die SKIP REST - Klausel wurde fuer eine Datei mit der Dateiorganisation STREAM verwendet (s. Abschn. 11.4.)

7401 Es wird versucht, das Protokoll einer Laufzeitprotokollierung ueber einen nicht aktiven Kanal auszugeben (s. Abschn. 12.2.).

7402 Es wird versucht, das Protokoll einer Laufzeitprotokollierung in eine Datei zu schreiben, die die Satzart INTERNAL oder die Zugriffsart INPUT besitzt (s. Abschn. 12.2.).

Ein- und Ausgabefehler

8001 Beim Lesen aus der internen Datenliste wird das Ende dieser Liste erreicht (s. Abschn. 10.1.).

! 8002 Bei Eingabe ueber Terminal enthaelt eine Eingabezeile zu wenig Daten (s. Abschn. 10.2.).
Behandlung: interaktive Fehlerkorrektur durch den Bediener

! 8003 Bei Eingabe ueber Terminal enthaelt eine Eingabezeile zu viele Daten (s. Abschn. 10.2.).
Behandlung: interaktive Fehlerkorrektur durch den Bediener

8011 Bei Abarbeitung einer Eingabeoperation wird das Dateiende erkannt (s. Abschn. 11.4.).

8012 Bei einer Eingabe (nicht ueber Terminal) enthaelt ein Datensatz zu wenig Daten (s. Abschn. 11.4.).

8013 Bei einer Eingabe (nicht ueber Terminal) enthaelt ein Datensatz zu viele Daten (s. Abschn. 11.5.).

8101 Beim Lesen aus der internen Datenliste wird als naechstes ein numerisches Datum gefordert, es liegt jedoch kein solches vor (s. Abschn. 10.1.).

! 8102 Die ueber Terminal eingegebene Eingabezeile ist syntaktisch nicht korrekt (s. Abschn. 10.2.).
Behandlung: interaktive Fehlerkorrektur durch den Bediener

! 8103 Bei der Eingabe ueber Terminal wird als naechstes ein numerisches Datum gefordert, es liegt jedoch kein solches vor (s. Abschn. 10.2.).
Behandlung: interaktive Fehlerkorrektur durch den Bediener

8105 Der aus einer Datei (nicht vom Terminal) gelesene Datensatz im Textformat ist syntaktisch nicht korrekt (s. Abschn. 11.4.).

8120 Bei der Eingabe von Daten im internen Format stimmen der Typ des Datums und der Typ der Variablen nicht ueberein (s. Abschn. 11.4.).

8201 Ein string-Wert, der als Formatstring verwendet werden soll, stellt keinen gueltigen Formatstring dar (s. Abschn. 10.4.).

8202 Im Formatstring ist kein Format enthalten (s. Abschn. 10.4.).

! 8203 Ein Format ist fuer die Ausgabe des betreffenden Wertes zu kurz (s. Abschn. 10.4.).
Behandlung: Das Ausgabefeld fuer den Wert wird mit dem Zeichen "*" gefuellt und der Satz ausgegeben. Die naechste Zeile enthaelt die nicht formatierte Darstellung des Wertes. Die Ausgabe wird in der Spalte der neuen Zeile fortgesetzt, in der sie bei korrektem Verlauf der Abarbeitung weitergefuehrt worden waere.

! 8204 Zur Ausgabe des Exponenten werden mehr Stellen benoetigt, als dafuer im Format vorgesehen wurden.
Behandlung: wie bei Ausnahme 8203

8301 Es wird versucht, einen Datensatz zu erzeugen, der laenger als die maximale Satzlaenge ist (s. Abschn. 11.3.).
8302 Es wird versucht, einen Datensatz zu lesen, der laenger als die maximale Satzlaenge ist (s. Abschn. 11.4.).

Geraetefehler (9000)

9000 nicht spezifizierbarer Geraetefehler

Steuerungsfehler (10000)

10001 Der Index in einer on-goto- bzw. on-gosub-Anweisung liegt ausserhalb des zulaessigen Bereiches und ein ELSE-Teil wurde nicht angegeben (s. Abschn. 8.2.).
10002 Es wird eine return-Anweisung ausgefuehrt, zu der es keine zugeordnete gosub- bzw. on-gosub-Anweisung gibt (s. Abschn. 8.2.).
10004 Bei der Abarbeitung einer Fallauswahl kann kein case-Block ausgewaehlt werden und ein case-else-Block wurde nicht angegeben (s. Abschn. 8.4.).
10005 Das durch eine chain-Anweisung gerufene Programm wird nicht gefunden (s. Abschn. 9.3.).
-10005 Eine gerufene externe Funktion bzw. Subroutine steht im Interpreter nicht zur Verfuegung.
! 10007 Eine break-Anweisung wurde abgearbeitet und der Teststatus ist aktiv (s. Abschn. 12.2.).
Behandlung: Dialog mit dem Bediener zum Ein- bzw. Ausschalten der Laufzeitprotokollierung und Programmfortsetzung